HUGHES
RESEARCH LABORATORIES

HAC REF. J6305

OPTICAL NEURAL NETWORKS BASED ON DISTRIBUTED HOLOGRAPHIC GRATINGS

Hughes Research Laboratories
3011 Malibu Canyon Road
Malibu, California 90265

August 1996

N00014-92-C-0187
Final Report
29 September 19992 through 28 February 1996

OFFICE OF NAVAL RESEARCH
800 North Quincy Street
Arlington, VA 22217-5660

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form ApprovedOMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Hughes Research Laboratories

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
Office of Naval Research

6c. ADDRESS *(City, State, and ZIP Code)*
3011 Malibu Canyon Road
Malibu, CA 90265

7b. ADDRESS *(City, State, and ZIP Code)*
800 North Quincy Street
Arlington, VA 22217-5660

8a. NAME OF FUNDING / SPONSORING ORGANIZATION

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
N00014-92-C-0187

8c. ADDRESS *(City, State, and ZIP Code)*
3701 North Fairfax
Arlington, VA 22203-1714

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNITACCESSION NO.

11. TITLE *(Include Security Classification)*
Optical Neural Networks Based on Distributed Holographic Gratings

12. PERSONAL AUTHOR(S)
Yuri Owechko

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM 9/30/92 TO 2/28/96

14. DATE OF REPORT *(Year, Month, Day)*
96 August 13

15. PAGE COUNT
92

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This final report describes research in optical neural networks performed at Hughes Research Laboratories under a three-year DARPA sponsored contract the advantages of optics for neural network implementations, including high storage capacity, connectivity, and very fine-grained parallelism, was demonstrated. The optical neurocomputer developed under this program is based on a new type of holography which we call multiple grating holography, in which this approach reduces crosstalk and improves the utilization of the optical input device. In addition, this optical neurocomputer is the first and, to the best of our knowledge, the only one which is programmable and capable of implementing a wide variety of neural network models without any hardware adjustments. Successfully implemented neural networks included the Perceptron, Bidirectional Associative Memory, Kohonen, and backpropagation neural networks. Up to 10^4 neurons, 2×10^7 weights, and processing rates of 10^8 connection updates per second were achieved. Under this contract, we built an optical neurocomputer which utilizes a laser diode light source operating at 830 nm. This allowed us to reduce the size of the system. We also developed a new method for representing bipolar neural weights using coherent detection.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
□ UNCLASSIFIED / UNLIMITED ☒ SAME AS RPT. □ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION

22a. NAME OF RESPONSIBLE INDIVIDUAL

22b. TELEPHONE *(Include Area Code)*

22c. OFFICE SYMBOL

Previous editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

CONTENTS

ILLUSTRATIONS

ILLUSTRATIONS

1. EXECUTIVE SUMMARY

This final report describes research in optical neural networks performed at Hughes Research Laboratories under a three-year DARPA sponsored contract (N00014-92-C-0187). This contract was a continuation of work performed under a previous three-year contract in which a programmable optical computer for flexible implementation of neural network models was designed, built, and demonstrated. The advantages of optics for neural network implementations, including high storage capacity, connectivity, and very fine-grained parallelism, was demonstrated. The optical neurocomputer developed under this program is based on a new type of holography which we call multiple grating holography, in which the neural network weights are distributed among cascaded angularly- and spatially-multiplexed gratings. This approach reduces crosstalk and improves the utilization of the optical input device. In addition, this optical neurocomputer is the first and, to the best of our knowledge, the only one which is programmable and capable of implementing a wide variety of neural network models without any hardware adjustments.

Under our first contract, we designed, built, and demonstrated an optical neurocomputer based on a water-cooled argon laser with a wavelength of 514 nm. This system was relatively large as it occupied most of a 4' × 5' optical table. Successfully implemented neural networks included the Perceptron, Bidirectional Associative Memory, and backpropagation neural networks. Up to 10^4 neurons, 2×10^7 weights, and processing rates of 2×10^7 connection updates per second were achieved. Under our second contract, which is the subject of this report, we built a second generation system which utilizes a laser diode light source operating at 830 nm. This allowed us to reduce the size of the system to the point where it fits on a 2' × 2' optical breadboard. Packaging concepts were formulated which would allow the system to fit in a 1' × 1' × 6" package using the same components as in the present system. We also developed a new method for representing bipolar neural weights using coherent detection. In addition to the neural networks mentioned above, we also implemented Kohonen's self-organizing map algorithm and applied it to handwritten digit recognition. Finally, we increased the learning/readout processing rate to 10^8 connection updates per second.

The organization of this final report is as follows. First, we briefly describe the nature of neural network models and the types of problems they are best suited to address. We describe real-time holography and its advantages and disadvantages for neural network implementations in terms of storage capacity, connectivity, and parallel processing. A new holographic technique, multiple grating holography (MGH), was developed by us to overcome an important source of distortion in holographic neural networks. We demonstrated MGH in photorefractive $BaTiO_3$ crystals using both visible light (514 nm) from an argon laser and infrared light (830 nm) from a laser diode. The infrared experiments are especially significant because very compact systems can be built using laser diode light sources.

We then discuss the design and construction of the optical neurocomputer based on MGH. The number of neurons, number of layers, and the neuron activation function can all be programmed without hardware adjustments. We believe ours is the first truly programmable optical neurocomputer. In addition, the simple system design requires only a single crystal, input spatial light modulator, and output detector regardless of the network configuration. The entire system was built from presently available off-the-shelf components. We also discuss packaging concepts in which the entire system is contained within a 1' × 1' optical breadboard.

2. NEURAL NETWORKS AND THEIR APPLICATIONS

The great success of conventional Von-Neumann computers and procedural programming techniques in solving algorithmic problems in science and engineering contrasts with the far slower progress in solving the ill-defined problems at which humans and animals excel, such as recognizing individual faces. It is readily apparent that the lowliest animal can perform feats of cognition which are beyond the abilities of the most powerful supercomputers programmed by the most talented computer scientists. The field of artificial neural networks has grown substantially in recent years as workers draw inspiration from biological nervous systems for approaches to solving such problems. In this final report we discuss a hardware architecture based on optical holography which is well-suited for efficient implementation of neural network models.

What distinquishes "easy" from "hard" problems for conventional procedural methods? Computational problems can be described in terms of many attributes. One of the most fundamental measures is the degree of "randomness" or algorithmic complexity of the problem. Algorithmic complexity can be defined as the length of the shortest procedural computer program that can generate a solution to the problem.[1] A completely random problem has very high algorithmic complexity since the only way to describe the solution is to literally list all the possible answers. This is analogous to the fact that the only way to describe a truly random number is to list all of its digits. No shorter algorithm exists.

By definition, nonrandom highly structured problems with low algorithmic complexity are best solved using traditional computer programs. In contrast, many problems involving natural data have a high degree of randomness, especially pattern recognition problems involving noisy data. These are problems that biological organisms excel at compared with classical algorithms in which rules are assumed *a priori* (before the calculation begins). Biologically inspired neural network models are useful for solving such highly complex problems in which the algorithmic solution is unknown and the required transformations must be learned from examples. They offer an alternative to standard linear and nonlinear statistical methods because, unlike other approaches, models of the data do not have to be assumed *a priori*, although prior information can (and indeed should) be incorporated if it is available. The nonlinear logical structure of neural networks also allows great flexibility in representing transformations.

In this final report we describe a particular hybrid *optical-electronic* architecture for implementation of artificial neural networks. The architecture is a good match for the requirements of neural networks because of the large storage capacity and parallel processing capabilities of optics. It takes advantage of a fortuitous match between the physics of holography in photorefractive crystals and a very general neural learning rule known as Hebb's Law. The architecture features a new recording method which eliminates an important source of distortion in holography. We have demonstrated several neural networks running on our optical neurocomputer. But first, let us discuss the structure and capabilities of neural networks.

2.1 Network Network Architecture

Neural network models of computation consist of many simple processing nodes or "neurons" which communicate with each other via interconnection weights. The nodes are called neurons in acknowledgement of the vastly more complex biological neurons which inspire neural network models of computing. The key word here is *inspire*. Only a very small fraction of the neural network models that have been proposed can be considered biologically relevant. Nevertheless, just as aircraft designs do not try to emulate birds in every way, so is faithfulness to biological accuracy of neural networks often relaxed in the interests of theoretical and physical practicality. In other words, although inspired by anatomical observations, the neural networks discussed here were conceived and developed in an engineering context. They should therefore be evaluated as engineering solutions to computational problems, not as accurate models of biological nervous systems. Knowledge of the actual information processing principles of biology is, apart from a few general principles, still largely unknown, although much progress has been made.

Many types of neural networks can be defined. The basic neural network structure is a directed graph with neurons at each node. The edges of the graph represent weighted connections between the neurons. Each neuron processes the weighted signals from other neurons, calculates a result, and passes it on to other neurons. The computation that the neural network performs is determined both by the topology of the graph and by the values of the connection weights. The weight values can be calculated "off-line" and loaded into the network, as in certain networks designed for optimization problems, or they can be "learned" by the network. (More on this later.)

Various classes of neural networks have been defined. They include random, recurrent, and laterally connected networks. In this chapter we will concentrate on the most widely-used topology: multi-layer feed-forward neural networks. A diagram of the logical structure of such a network is shown in Figure 1. The neurons are arranged in layers with connections between layers only. Associated with each neuron i in layer n is an activation level y_i^n which is calculated by passing a weighted sum of the activation levels of a subset of other neurons in the previous layer through a nonlinearity f:

$$y_i^{(n)} = f\left(x_i^{(n)} - \theta_i\right)$$

$$x_i^{(n)} = \sum_j w_{ij}^{(n)} y_j^{(n-1)} \qquad (1)$$

where θ_i is an adjustable threshold value. The nonlinearity f is typical a "sigmoid" function such as

$$f(x) = \frac{1}{1+e^{-x}} \qquad (2)$$

which is monotonically increasing and limited between 0 and 1 (or between -1 and 1). Patterns which are input to the network through the bottom layer are transformed into top layer patterns which represent the result of the computation. The "program" or instructions for the computation is encoded in the structure of the network (interconnection pattern) and the weight values, w_{ij}, of the interconnections.

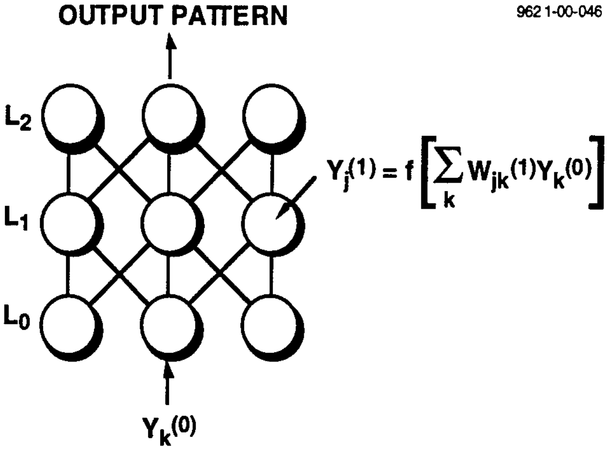


Figure 1. Structure of feed-forward neural networks.

Networks with a single layer of weights can only solve "separable" problems for which a solution can be obtained by separating disjoint classes of patterns with a hyperplane in pattern space.[2] However, by adding a "hidden" neuron layer with enough neurons, any nonlinear transformation can, in principle, be represented by a neural network.[3] Let us define "pattern space" as a space where each coordinate represents the activity level of an input neuron. Then an input pattern defines a single point in pattern space. The hidden layer allows the network to divide pattern space into disjoint regions using combinations of hyperplanes. These regions of pattern space can be logically combined by the output layer in order to form the final result of the calculation. It is sometimes worthwhile to add a second hidden layer in order to improve performance.

Implementing large neural networks while maintaining computational parallelism is a daunting task for electronic architectures due to the 2-D nature of electronic interconnects. Most of the area on analog or digital electronic neuro-chips is taken up by the interconnects while implementing only moderate numbers of neurons. Optical implementations of neural networks are attractive because of the large storage capacity and, most importantly, the parallel access and processing capabilities of optics. Optical architectures can exploit 3-D free space interconnects, allowing the input and output planes to be fully populated with highly interconnected neurons ($N=O(10^5)$). Moreover, an entire weight layer can be updated in one time step. The optical neurocomputer described in this chapter uses 3-D weight storage based on volume holography. The primary motivation for considering volume holograms as a storage medium for neural networks is the potential for extremely high storage capacity combined with fully parallel processing of the weights during both the learning and reading phases.

Of course, an important issue is how to find the proper weight values for solving a particular problem. Although no general method has been discovered, many neural net architectures have been designed and demonstrated for specific applications. Various techniques have been developed for "learning", which may be defined as the non-algorithmic adjustment of connection weights to solve problems in such fields as pattern recognition, vision, and robotic control. Issues related to learning are discussed in the next section. But first, let us consider the requirements for neural network hardware imposed by various applications.

Figure 2 is an adaptation of a figure which originally appeared in the final report of the 1988 DARPA Neural Network Study. It shows the potential application areas of neural networks mapped onto a 2-D space in which one axis is the storage required in terms of the number of weighted connections and the other axis is the processing rate required in connections per second. The corre-

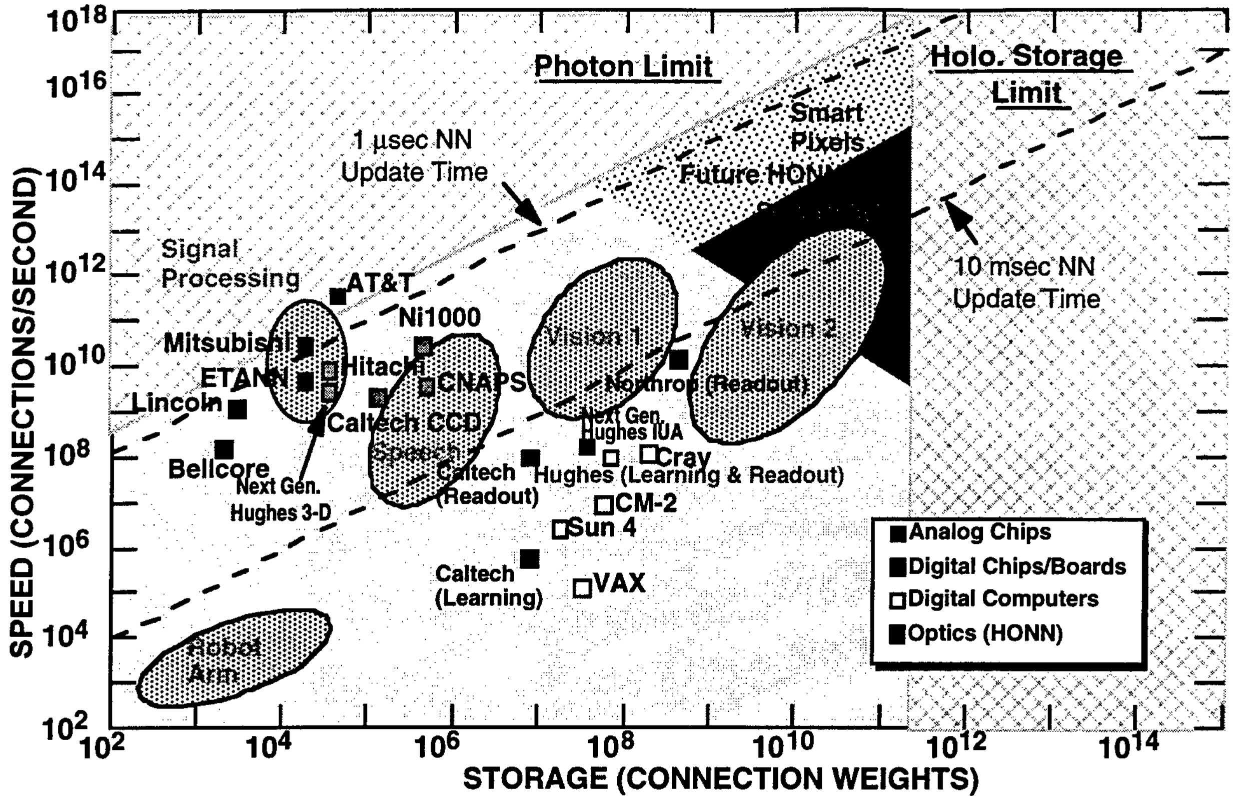


Figure 2. Neural network hardware performance parameters and requirements of various application areas. *Electronic and optical technologies tend to be complementary in their capabilities. Optics is most well-suited for implementing large neural networks with high connectivity and storage requirements.*

sponding estimated parameters of some biological systems are included. A variety of recent electronic implementations, denoted by open squares^4, have also been added to Figure 2. The electronic implementations are, apart from the Sun and Cray computers, specialized analog and digital chips. Not all of the neuro-chips implement learning. (For non-learning chips, weight values must be determined by other means and then loaded in.) Finally, diagonal lines of constant network update time were added to the figure. The update time is the time required for information to pass from the input of the network to the output. Of course, such a plot is necessarily a highly folded and simplified projection of a multi-dimensional reality onto a two-dimensional graph. For example, the degree of local vs global connectivity is ignored as well as the algorithmic complexity of the application. In some cases every neuron is connected to every other neuron while in others the neurons may be sparsely interconnected. Also, in some hardware implementations the learning rate is much slower than the readout rate. Nevertheless, with proper caution we can observe certain trends.

It is interesting that the potential application areas in robotics, speech, and vision cluster around the 10 msec network update time line. These applications require large numbers of weights in order to achieve the complexity required to solve the problem, but the solution does not need to be obtained in less than a few milliseconds. The large number of weights, however, requires very high processing rates to achieve this update time. Interestingly, the biological systems also cluster around the 10 msec update line, reflecting the typical time constants of neural signals. With the exception of the general purpose computers, the electronic implementations have relatively modest storage capacity, although their processing rates are high. (Ignoring the fact that many of the chips do not have on-chip learning.) This is due to the 2-D nature of VLSI which limits the number of connections that are practical. They therefore appear most suited to signal processing applications in which fast update times and modest storage are required. (For example, such chips will soon appear in TV sets implementing noise reduction and line interpolation functions. The neural chip learns to form a better and cheaper filter than current designs.)

In our view, optical neurocomputers are complementary to electronic versions in that the 3-D connectivity and parallelism of optics permit the implementation of very large networks with high processing rates and relatively modest network update times. The projected future performance of holographic optical neural networks (HONN) is indicated by the region marked “Future HONN” in Figure 2, a level of performance beyond that forecast for electronic architectures. HONN performance is bounded by two fundamental physical limits. The photon limit limits how short the network update time can be since a minimum number of photons integrated over the update time is required for an adequate detector signal to noise ratio. The holographic storage limit determines the maximum number of weights that can be stored. We predict that the highest performing HONNs will be based on smart pixel technology, as opposed to SLMs (spatial light modulators). In order to fullfill their potential and find market acceptance, such optical neurocomputers should include, first, large numbers of neurons and weights with programmable connectivity; second, distortionless software mapping of a variety of neural network algorithms without requiring hardware reconfiguration; third, co-processor-type interfacing to a host electronic computer; and fourth, hardware simplicity for low cost and compact packaging.

In Section 4 we describe an experimental optical neurocomputer which serves as a testbed for meeting these requirements. The neurocomputer is a nonlinear, highly interconnected, parallel, and analog opto-electronic computer based on real time holography. The present performance of our second generation laboratory prototype developed under this contract is indicated by the filled square labeled “Hughes” in Figure 2. Two other proof-of-concept holographic neural networks are also shown in Figure 2. The Caltech HONN is a single-hidden-layer network that has been used to recognize human faces.[5] Because each hidden neuron was trained sequentially, the learning speed was much less than for readout. The Northrop HONN is an inner product processor which correlates an input image with 5000 separate filters simultaneously.[6] The storage and readout performance figures are impressive, and prove the capabilities of holographic storage. However, learning was not implemented and the time required to load the weights into the holographic crystal was relatively long, again because each filter was recorded sequentially. Unlike our prototype, the other HONNs are not programmable and were built to implement a specific neural network.

Before we delve into the details of holography and optical neurcomputers, let us first continue our discussion of neural networks by exploring what may be their most fascinating feature: the ability to “learn” how to solve problems.

2.2 Learning in Neural Networks

What do we mean when we say neural networks can "learn"? We mean that by providing examples of a function operating on input patterns, and by adjusting the weights using a learning algorithm, a neural network can converge to an approximation of the function. In theory, the function can be quite general, including, for example, such tasks as recognizing a face or controlling a robotic arm. If the approximation is a good one, new input patterns that were not part of the training set will also be correctly processed by the network, in the sense that the network outputs will be close to those of the function being modeled. In this respect the network has learned to "generalize" from a finite set of exemplar transformations by properly processing novel patterns that are not members of the training set. The network does this by extracting common features in the exemplars and encoding them in the weights. Although the general problem of learning an arbitrary transformation for a completely random problem has been shown by Judd[7] to be a member of the NP-complete class of problems (therefore probably requiring exponential increases in learning time as the problem size increases), in practice specific problems in pattern recognition involve at least partially structured data for which the learning time can often be reduced to a polynomial function of problem size.

Some neural models such as the Hopfield network don't learn in the sense discussed above. Instead, the weight values are calculated according to a given algorithm or recipe. The multilayer perceptron using backpropagation, on the other hand, is an example of *supervised learning*. Using an error signal generated by an external "teacher", the network is trained to minimize the differences between the exemplar and network output patterns. In *unsupervised learning*, input exemplars are presented to the network without external guidance as to what the correct output should be. The network then clusters them into self-similar classes. Examples of this kind of network are Grossberg's Adaptive Resonance Theory[8] networks and Kohonen's Self-Organizing Maps.[9]

Neural networks can be viewed as pattern recognizers or classifiers. It is therefore natural to compare them with statistical pattern recognition techniques. As in neural network learning, the goal of classical statistical pattern recognition is to estimate the *a posteriori* probability P(ω_i|**x**) that a given pattern **x** belongs to class ω_i. In statistics, *a posteriori* probabilities are calculated from observations while *a priori* probabilities are determined from past knowledge or measurements. Once we know P(ω_i|**x**) the problem is solved: the correct class is determined by the largest P(ω_i|**x**). This procedure is the optimum one in a statistical sense and is known as the Bayes optimum discriminant function. Unfortunately, the *a posteriori* probabilities and discriminant functions derived from them are generally unknown. The methods of classical statistical pattern classification are to a large degree concerned with the use of Bayes Rule to estimate *a posteriori* probabilities from measurements of *a priori* probabilities. Bayes Rule is

$$P(\omega_i \mid x) = \frac{P(x \mid \omega_i)P(\omega_i)}{P(x)} \tag{3}$$

where P(**x**|ω_i) is the *a priori* probability of pattern **x** occuring given class ω_i, P(ω_i) is the *a priori* probability of class ω_i, and P(**x**) is the *a priori* probability of pattern **x** without regard to class membership. Using the definition of conditional probabilities,

$$P(x) = \sum_j P(x \mid \omega_j) P(\omega_j) \tag{4}$$

Ruck et al.[10] have shown that neural networks can be trained to directly give unbiased estimates of the *a posteriori* probability $P(\omega_i|\mathbf{x})$. In fact, they showed that the outputs approach $P(\omega_i|\mathbf{x})$ for any neural network with outputs between 1 and 0 which is trained to reduce the mean-square error for the exemplar set. The outputs of a network can therefore provide a confidence measure for classification. Of course one still has to choose a neural network structure with a set of parameters suitable for modeling the probabilities.

The size and structure of a neural network has a great effect on its generalization ability. In general, too small a net won't be able to model the problem, but too large a net will not generalize well. This is because overly large networks with too many degrees of freedom can find solutions that are consistent with the exemplar set but which are not at all a general solution to the problem. For example, if the number of neurons in the hidden layer of a multilayer perceptron is much larger than the number of training exemplars, then the net has enough degrees of freedom to find specialized solutions which agree with the exemplars but do not generalize to new inputs. Such a situation is analogous to the overfitting of a low-degree curve with a high degree polynomial: the original data points can be fitted very well but new points can lie far from the curve. Good generalization depends on finding the *simplest* neural network which correctly processes the exemplar set.

Although no theoretical result exists that prescribes the best network structure for a given specific problem, recent work in learning theory has resulted in some theoretical bounds that can provide guidance for designing neural networks that will generalize well. Let's assume we wish a neural network to learn a particular Boolean function F(**x**) from a set of p exemplars. The exemplars are drawn from a class of patterns, C, with common properties. If $f_w(\mathbf{x})$ is the function that the network actually implements for a given set of weights, then we wish to find the generalization $g(f_w)$, which is defined to be the probability that $f_w(\mathbf{x})=F(\mathbf{x})$ for pattern **x** randomly drawn from C. Let's also assume that, after learning, a fraction $g_p(f_w)$ of the exemplar set is correctly processed by the network. The question we would like to answer is how well does the network perform on patterns which are members of C but not of the exemplar set? In other words, how close is $g(f_w)$ to $g_p(f_w)$? We cannot blindly assume $g(f_w)=g_p(f_w)$ because $g_p(f_w)$ may be strongly biased towards the exemplars (the overfitting problem).

By considering how many different functions a given neural network can implement, Vapnik and Chervonenkis[11] proved an upper bound to the difference between $g_p(f)$ and $g(f)$ for any possible function f implementable by the network:

$$\text{Prob}\left(\max_f \left| g_p(f) - g(f) \right| > \varepsilon \right) \le 4m(2p)e^{-\varepsilon^2 p/8} \tag{5}$$

The growth function m(p) is defined to be the number of different binary functions that can be implemented by the network on any set of p input patterns. It is clear that m(p) cannot be more than the number of possible binary functions of p points, which is 2P. The left hand side is the probability that the estimate of generalization, based on the p exemplars, differs from the actual generalization by more than ε. In particular, if we can show that the right hand side is small for small ε, then we can be sure that if the performance on the exemplar set is good then with high probability the performance will also be good for new patterns drawn from the same distribution.

Vapnik and Chervonenkis proved that m(p) always looks as shown in Figure 3. It is equal to 2P up to the point p=d_{VC}, where the growth starts to slow down. d_{VC} is called the Vapnik-Chervonenkis dimension, or VC-dimension. It is a function of the structure of the network and the number of neurons and weights. Clearly for the best generalization we should use a neural network with the smallest d_{VC} which is still capable of representing solutions to the problem at hand. In general, a given neural network structure cannot implement any arbitrary binary function. This implies that d_{VC} is finite for most networks. It can be shown that if d_{VC} is finite, then m(p) obeys the inequality

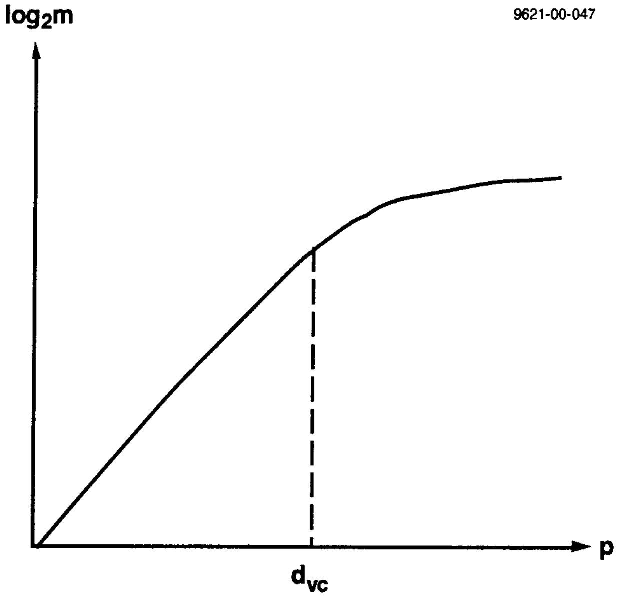


Figure 3. General form of the growth function m(p), which is equal to 2^p for p less than d_{vc} and then bends over.

$$m(p) \le p^{d_{vc}} + 1 \tag{6}$$

which can be plugged into the above expression in order to obtain an upper bound for the generalization error. d_{VC} has been estimated for several different networks. For example, single-layer Perceptrons can only implement separable dichotomies of patterns so for that class of networks d_{VC} is equal to the number of input neurons.[12] Baum and Haussler calculated upper and lower bounds to d_{VC} for multi-layer feed-forward networks.[13] They found that d_{VC} is upper bounded by

$$d_{VC} \le 2W \log_2(eM) \tag{7}$$

where $\lfloor x \rfloor$ is the largest integer not greater than x. Taken together, these results indicate that in general one needs of the order of W/ε exemplars in order to obtain a generalization error less than ε.

Based on the above results, a practical rule of thumb for good generalization is that the number of exemplars should be a few times larger than the the number of weights in the network. Bear in mind, however, that this estimate comes from a worst case analysis. For any particular problem, it may be possible to use fewer exemplars, especially if *a priori* knowledge about the solution can be incorporated in the network structure. Various techniques have been developed to reduce d_{VC} by pruning unimportant weights, using local connections, sharing weights, or allowing weights to decay.[14] Another method to reduce d_{VC} by incorporating prior information is to provide "hints" to the network.[15] For example, translational invariance can be built into the network structure so that it doesn't need to be learned.

An important parameter of neural networks is the connectivity, K, which is the number of synapses or weights connected to a neuron. Abu-Mostafa has shown that K should be large for at least two reasons.[16] First, since the neurons essentially implement K-input threshold functions, one K-input neuron with K associated weights is equivalent to order K^2 two-input neurons with $2K^2$ associated weights. Thus far fewer weights are required if the neurons have high fan-in and fan-out. Second, he showed that, in order for a neural network to learn, the connectivity K must exceed the entropy H of the environment where H is the $\log_2$ of the number of input patterns typically generated by the environment. H increases as the randomness of the problem increases.

2.3 Application of Holographic Neural Networks

As discussed in Section 2.1, the most likely market entry points for HONN are applications which require large neural networks to represent the solution to complex problems. The computation rate must also be high in order to obtain a solution in a reasonable amount of time. The following list is not meant to be inclusive, it only serves to illustrate the types of applications that could benefit from commercialized HONN technology.

Machine Vision. The problem of automated recognition of objects in an input scene is an important one and its solution would have major beneficial effects on industrial productivity and the economy. The general problem of recognizing 3-D objects from 2-D projections of a scene is under-constrained and hence intractable without incorporating additional knowledge. By demonstrating just how difficult object recognition really is, research in machine vision has verified the remarkable computational power of the biological vision system. Practical machine vision systems must greatly restrict the domain of possible input scenes in order to be successful. Applications for machine vision include cursive handwriting recognition for signature verification, automated note transcription, and Postal Service mail sorting; segmentation of satellite imagery; industrial part inspection; and target recognition for terminal guidance of missiles, among others. Many of the most successful approaches to these problems are based on neural networks or use neural networks as at least one element of the processing chain. Neural networks can be used both in the "front end" for extracting primitive features from the raw data and in the "back end" for data fusion, classification, and final identification of objects.

Input scenes typically contain many pixels, from hundreds to hundreds of thousands. Neural networks are intrinsically parallelizable, but processing a scene in parallel would require such large numbers of neurons that electronic implementations must scan the image at a large cost in processing time. The parallelism and high storage capacity of HONN architectures allow much greater numbers of neurons to be implemented, thus making the parallel processing of input scenes tractable. The parallelism of HONN is also flexible: many feature maps can be generated simultaneously for classification of multiple objects in a scene.

A generalization of neural networks, called "higher order" neural networks, has many theoretical advantages for machine vision.[17] In higher order neural networks, weights are formed not between neurons, but between products of neuron values. The weight matrix then becomes a weight tensor. For example, the output of a layer in a third-order neural network is given by

$$y_i = f\left(\sum_j \sum_k \sum_l w_{ijkl} x_j x_k x_l\right) \tag{8}$$

The usefulness of higher-order neural networks lies in the fact that various invariances can be built into the network. For example, a third-order neural network can be made to be invariant to rotation, scale, and translation shifts of objects in the scene by forcing the weights to be of a particular form. The network then responds only to groupings of triangles. Similar triangles with the same interior angles evoke identical responses, the network is then automatically invariant to scale, rotation, and translation shifts. Such invariances are obviously useful for machine vision. Higher-order neural networks have not been widely used because of the large increase in weights that occurs when connections are made between products of neuron values. The ability of HONN to implement large networks may be an enabling technology for the application of higher-order neural networks to machine vision problems.

Speech Recognition. Most current state-of-the-art continuous speech recognition systems are based on hidden Markov models. A hidden Markov model (HMM) is a stochastic finite state machine. At any one time the HMM may be in one of a set of "states" that are hidden from direct observation. A transition probability distribution describes the time evolution of the HMM as it evolves from state to state. The HMM is also characterized by an observable output symbol which is described by an observation probability distribution. In speech recognition systems the observation probabilities are used to model time-localized speech features (phonemes) and the transition probabilities model the time evolution of the phonemes.

Neural networks have been used to significantly improve the performance of HMM-based speech recognition systems by estimating the HMM observation probabilities from training data. Moreover, the context-dependence of phoneme structure can be modeled. In speech the spectral content of a phoneme depends on the phonemes that come before and after it. Neural networks can incorporate this contextual information in the estimated observation probability distributions. The combined neural network-HMM results in superior performance and a reduction in the number of model parameters compared to context-dependent HMMs. Because of the large number of possible phoneme features and contexts, large neural networks are used in the speech recognition application. For example, one group has used a neural network with 1.4 million weights in a context-dependent neural network-HMM speech recognition system.[18] The size of the network was limited by the special purpose electronic hardware used in the training phase. A HONN would allow much larger neural networks to be implemented, thereby increasing the number of contexts and further improving the speech recognition system performance.

These are only some example applications of neural networks. Other applications include financial analysis, associative retrieval of information, optimization problems, and process control.[19]

3. HOLOGRAPHIC IMPLEMENTATIONS OF NEURAL NETWORKS

3.1 Basic Architecture

Optical neural networks divide naturally into two classes distinquished by the dimensionality of the weight storage medium. Weights can be stored in 2-D or 3-D formats. Example 2-D weight storage media include film, SLMs, and optical disks while almost all of the 3-D formats use photorefractive crystals as the storage medium. The optical processor described here uses 3-D weight storage based on volume holography. The primary motivation for considering volume holograms as a storage medium for neural networks is the potential for extremely high storage capacity and fully parallel processing of the weights in both the learning and reading phases.[20]

In holographic optical neural networks, neurons are represented by pixels on spatial light modulators (SLMs). For our purposes an SLM can be described as an adjustable transparency consisting of a 2-D arrangment of pixels which is controlled by a host computer. The transmission of a pixel corresponds to the activation level of the neuron. By placing the SLM in the back focal plane of a lens and using coherent readout, as shown in Figure 4, the pixels are converted to coherent beams which illuminate a real-time holographic medium. Weights between neurons are formed when a pair of light beams interfere in the holographic medium, forming a volume sinusoidal light intensity pattern. The photorefractive effect is a suitable physical mechanism for converting this light intensity pattern into a semipermanent deformation of the optical properties of the material, thereby recording the weight values. A variety of optical neural networks based on holography have been proposed or demonstrated.[21-23]

In the photorefractive effect incident light excites charges into the conduction or valence band. These charges then are transported by diffusion and drift until they fall into empty traps, creating an internal space-charge field which in turn modulates the birefingence of the material through the electro-optic effect. This results in a phase grating in the material. Because of the long dark decay times of some of these materials, the phase gratings can be stored with a time constant of many hours. (Storage for longer periods is also possible in some materials using various fixing methods.) When one of the original two beams subsequently addresses the grating, the other beam is reconstructed.

The diffraction efficiency of the semi-permanent phase grating represents the connection weight formed between the neurons that wrote the grating. It is proportional to the outer-product of the amplitudes of the writing beams and has a fortuitous similarity to the Hebbian learning rules of many neural network models. This equivalence between the outer-product form of the diffraction efficiency and neural network learning forms the basis for implementing the weights directly using the analog laws of physics rather than digital representations as in conventional computers. Learning can be implemented in photorefractive optical neural networks since the weights can be selectively increased or decreased. Reading out the grating partially erases it unless the readout beam is much weaker than the original writing light or the crystal is fixed using special techniques.

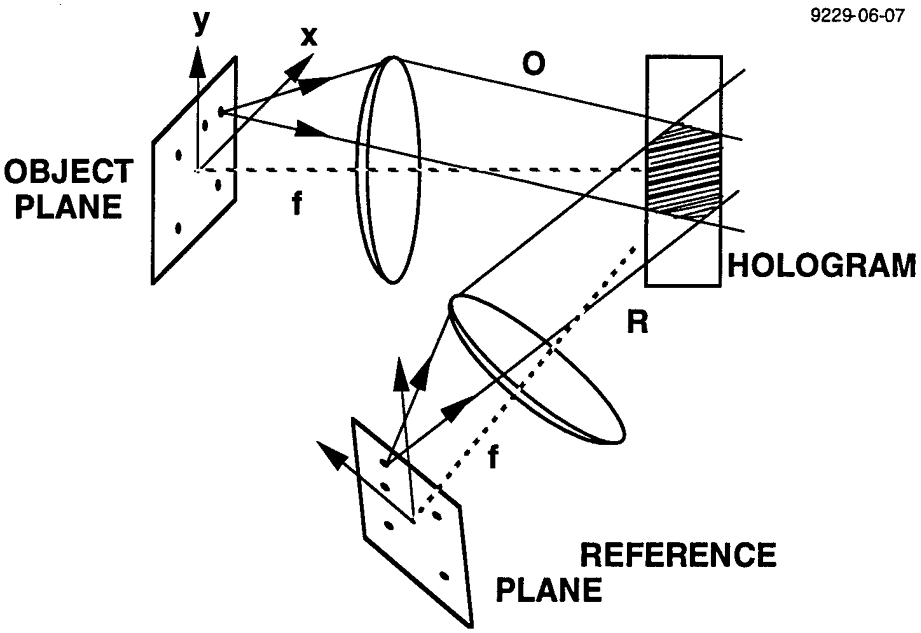


Figure 4. Geometry for recording connection weights between neurons holographically.

The angular or Bragg selectivity of a volume photorefractive grating can be very high which results in large storage capacities. The Bragg condition states that a beam will be reconstructed only if the angle of incidence of the incident beam relative to the grating is approximately equal to that of the original writing beam. The angular selectivity for reconstruction can be calculated from coupled mode theory[24] and is given by

$$\Delta\theta = \frac{\lambda}{nT_z \sin\varphi} \tag{9}$$

where T_z is the grating thickness, λ is the optical wavelength, n is the index of refraction, and ϕ is the angle between the two writing beams. Note that the selectivity is greater for thicker crystals. Each individual light beam can be represented by a wavevector or k-vector. (The direction of the k-vector corresponds to the direction of propagation and the magnitude of the k-vector is the inverse of the wavelength.) By using phase matching arguments, the Bragg condition can be described geometrically as a vector sum: $\mathbf{K_j} + \mathbf{K_g} = \mathbf{K_i}$, where $\mathbf{K_j}$ and $\mathbf{K_i}$ are the wavevectors of the incident and diffracted beams, respectively, and $\mathbf{K_g}$ is the grating wavevector.

A geometrical construction for the theoretical storage capacity of a volume hologram can be drawn in grating k-space, as shown in Figure 5. If one writing beam varies over solid angle θ_o while the second writing beam varies over angle θ_r, then the vector difference between the two beams (the grating wavevector K_g) will trace out a three-dimensional region in k-space (shaded gray in Figure 5). This volume represents the region of k-space that is accessable for storage of information. It will be further limited by the resolution or modulation transfer function (MTF) of the holographic medium. This limit is represented by a sphere centered on the origin whose radius is equal to the

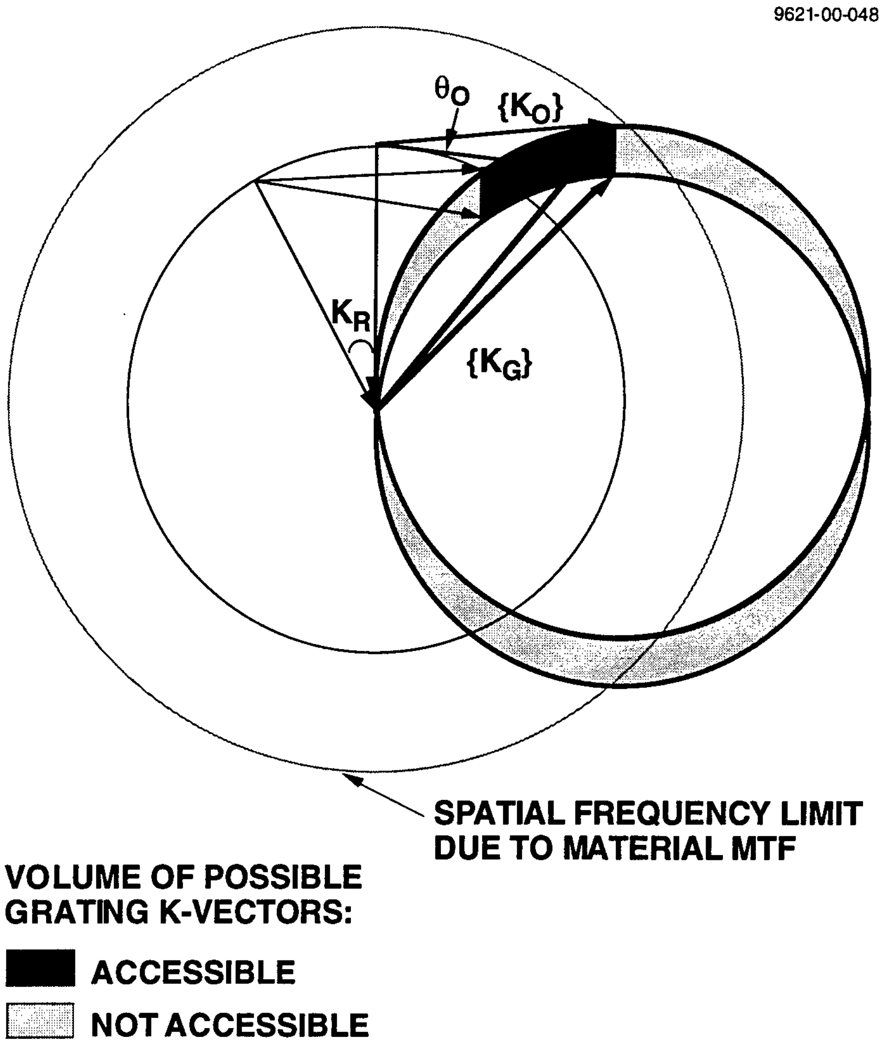

9621-00-048
θ_O
$\{K_O\}$
K_R
$\{K_G\}$
SPATIAL FREQUENCY LIMIT DUE TO MATERIAL MTF
VOLUME OF POSSIBLE GRATING K-VECTORS:
■ ACCESSIBLE
▨ NOT ACCESSIBLE

Figure 5. K-space construction of grating k-vectors accessable using recording geometry of Figure 2.

largest spatial frequency that can be resolved by the medium. The volume of the accessable region depends on such geometrical factors as the focal lengths of the optics and the spacing of the neurons on the SLMs.

The grating wavevector $\mathbf{K_g}$ has an uncertainty volume associated with it due to the finite physical size of the hologram and the nonzero size of the SLM pixels. Dividing the accessible volume of k-space by the volume of the uncertainty volume results in the maximum theoretical number of resolvable gratings which can be stored in the photorefractive crystal. Without going into details here it can be shown that the storage capacity is limited by two upper bounds due to the hologram and neuron dimensions:

$$No.of\ connections \propto \left\{ \begin{matrix} f^3/d^3_{pixel} \\ V_{holo}/\lambda^3 \end{matrix} \right\} \propto N^3 \qquad (10)$$

where V_{holo} is the hologram volume, f is the lens focal length, and d_{pixel} is the SLM pixel diameter. It was assumed in the calculations that led to the above result that the hologram MTF limit is high enough to be ignored. For an active crystal volume of a few cubic mm and reasonable optical parameters the values of the two upper bounds range from 10^{10} to 10^{12} gratings. This is sufficient to form a fully interconnected network of 10^5 to 10^6 neurons. Moreover, each grating can be read out or updated in parallel without the time multiplexing, data contention, or bottleneck problems common in electronic architectures. As of this writing, up to 5000 high quality images consisting of 70,000 pixels each have been stored in a single photorefractive crystal.[25] This corresponds to a demonstrated storage capacity of 4×10^8 connection weights, which is still well below the above estimates of the achievable storage capacity.

The learning phase of neural network models involves many parallel weight adjustments in response to an internally or externally generated error signal. In holographic optical neural networks this function is performed through the superposition of many exposures of the photorefractive crystal in which the strengths of individual gratings are strengthened or weakened. Ideally, the holographic process should not distort the linear superposition of weight update vectors in order to faithfully implement neural network models. Photorefractive crystals differ from recording-center-based holographic materials, such as film, in that gratings are recording by the optical redistribution of a fixed amount of charge. Thus, as new gratings are written, old gratings are necessarily erased in a process called *weight decay*. In the following we point out how weight decay can distort the neural network learning rule if precautions are not taken. In addition, we show that weight decay also reduces the magnitudes of weight update values as the number of training exposures increases, which will ultimately limit the number of exposures that can be recorded.

If we assume a simplified approximation for the photorefractive equations developed by Kukhtarev et al.,[26] in the absence of an applied field, the time dependence of the space charge field E_{ij} written by two plane waves with amplitudes A_i and A_j can be expressed by

$$\frac{\partial E_{ij}}{\partial t} = -\frac{I_0}{\tau_1} E_{ij} + \frac{iE_{sc}A_iA_j^*}{\tau_1} \qquad (11)$$

where $i = \sqrt{-1}$, $I_0 = \Sigma_i |A_i|^2$ is the total optical intensity, and τ_1 and E_{sc} are material constants. The space-charge field grating, E_{ij}, which arises from the diffusion of optically-excited charges away from regions of high light intensity, represents the holographic interconnection between optical beams A_i and A_j. Due to the diffusion process, the space-charge grating is shifted by 90 °relative to the optical grating. It is important to appreciate that a set of connections between many beams can be formed in parallel, each representing an independent connection weight.

Let us rewrite Eq. 11 using neural network terminology. Each weight w_{ij} connecting neurons i and j is then described by an equation of the form

$$\frac{dw_{ij}}{dt} = -\frac{w_{ij}}{\tau} + \eta_{ij} \qquad (12)$$

where $\eta_{ij}=iE_{sc}A_iA_j^*/\tau_1$ is the updating term proportional to the product of the amplitudes of writing beams i and j. The time constant τ is equal to τ_1/I_0, it decreases proportionally as the total light intensity increases.

The solution to the above differential equation has a simple exponential form:

$$w_{ij} = \eta_{ij}\tau + \left(w_{ij}^0 - \eta_{ij}\tau\right)e^{-t/\tau} \tag{13}$$

where w^0_{ij} is the initial or starting value of the weight. The saturation or steady-state weight value for $t \gg \tau$ is given by

$$\begin{aligned} w_{ij}^{sat} &= \eta_{ij}\tau \\ &= iE_{sc}A_iA_j^*/I_0 \end{aligned} \tag{14}$$

The steady-state weight is normalized by the total light intensity, its value is determined by the relative modulation depth of the optical grating. In photorefractive materials, higher optical powers speed up the time response but do not change the final diffraction efficiency. For exposure times short compared to τ, Eq. 12 yields a simple expression for the change in w_{ij} caused by a short exposure Δt:

$$\Delta w_{ij} \equiv w_{ij} - w_{ij}^0 \approx \left(\eta_{ij} - \frac{w_{ij}^0}{\tau}\right)\Delta t \tag{15}$$

Almost all neural network models use *Hebbian* learning rules for which Δw_{ij} is proportional to η_{ij}. From the above equation we can see that weight modification is slightly different in holographic neural networks. The Hebbian change in w_{ij}, $\eta_{ij}\Delta t$, is modified by the additional term $-(w^0_{ij}/\tau)\Delta t$ which represents the effects of weight decay. The relative influence of the weight decay term is smaller for smaller weight values. Let us now explore how weight decay affects photorefractive neural networks.

First, define a weight vector $\mathbf{w_i}$ as the set of weights which funnel signals into neuron i. If $\eta_{ij}=0$ ($A_i = 0$) for a particular weight vector, then we have pure weight decay and the weight vector's *magnitude* is reduced but not its *orientation.* This is due to the change in each weight being proportional to the weight value. The orientation of a weight vector is, however, also affected if η_{ij} is nonzero (e.g. both A_i and A_j are nonzero). The relative effects of weight decay on learning are small for small weights which is normally the case during the initial stages of learning. The decay term can distort gradient descent neural network models such as backpropagation which are sensitive to both the orientation and magnitude of $\Delta\mathbf{w_i}$. Other neural networks, such as the Perceptron, are relatively insensitive to weight decay. As previously mentioned, weight decay can be reduced by using small weight values, which corresponds to low diffraction efficiency holograms. On the other hand, as mentioned in Section 2.2, weight decay can also be useful by improving the generalization performance of the network.

We will now show that if the individual exposures are short enough, the final weight is simply proportional to the linear average of the update values. This recording method is known as *incre-*

mental recording[27] as opposed to *scheduled recording*[28] where the update values are equalized by using different exposure times for each hologram. Incremental recording is more suited to neural network models because the total number of exposures does not need to be known before learning can commence.

Using the simple exponential exposure model introduced above, it can be shown that after a sequence of N exposures, each of which lasts T seconds, the final weight value is given by

$$w_N = \tau \left(1 - e^{-T/\tau}\right) \sum_{p=0}^{N-1} \eta_{N-p} e^{-pT/\tau} \tag{16}$$

where in the interest of clarity the connection indices i and j have been suppressed and the subscripts are now exposure indices. In order to minimize distortions due to photorefractive weight decay, w_N should be forced to be proportional to η_n averaged over the N exemplars. This occurs if the individual exemplar exposure times are short compared to τ:

$$w_N \approx \left(1 - e^{-NT/\tau}\right) \overline{\eta} \tau \quad \text{for} \quad T \ll \tau \tag{17}$$

where

$$\overline{\eta} = \frac{1}{N} \sum_{n=1}^{N} \eta_n$$

If the number of exposures is large then

$$w_N \approx \overline{\eta} \tau \qquad \text{for } N \gg \tau/T \tag{18}$$

Thus under these conditions the weight is simply proportional to the linear average of the update values. The diffraction efficiency, which is proportional to $|w_N|^2$, decreases as $1/N^2$. This is different from conventional neural network models in which the final weight value is given by a simple sum of update values, not averaged over the total number of updates. In photorefractive neural networks, the absolute value of each weight update decreases as the total number of exposures increases. Eventually, for large numbers of exposures, each weight update will become smaller than the background noise. Thus, weight decay will limit the total number of exposures that can be recorded in a photorefractive crystal to a finite value which is dependent on the dynamic range of the holographic gratings. Calculating this limiting value requires formulation of a noise model which is beyond the scope of this chapter. However, recent experimental measurements indicate that the dynamic range of photorefractive gratings exceeds 100 db, or a factor of 10^{10} in diffracted light intensity.[29] This measured value was limited by detector noise. Theoretical calculations result in an upper bound of 140 db for the grating dynamic range.[30] Since the diffraction efficiency decreases as $1/N^2$, the maximum number of exposures that could be stored would be limited to less than 10^7.

It is informative to contrast the implications of Eq. 12 for holographic data storage and neural network applications. If each exposure creates an independent hologram without gratings common to other holograms, as is usually the case for holographic data storage where multiple data pages are stored in the same crystal volume using angle or wavelength multiplexing, then each weight has only one nonzero η_n associated with it. For independent superimposed holograms, the intensity diffraction efficiency, $|w_Q|^2$, is then given by

$$\left|w_Q\right|^2_{independent} \approx \left(\frac{\eta\tau_E}{N}\right)^2 \tag{19}$$

where η is the nonzero value of η_n. Each individual hologram's intensity diffraction efficiency is decreased by the square of the total number of holograms.

In photorefractive neural networks on the other hand, the superimposed holograms are not independent: many gratings are common to multiple weight updates. In a neural network, many if not most of the weights are continually adjusted during the learning procedure. Thus each grating may be updated many times as opposed to once if the holograms were independent. The effects of grating erasure are therefore less than would be the case for independent holograms. Instead of $\overline{\eta}$ being equal to the one nonzero η_n divided by N, for non-independent holograms it is equal to the sum of many nonzero η_n's divided by N. For example, assume that the weight w_Q is updated by the nonzero value η in a fraction ε of the N exposures. Then

$$\overline{\eta} = \frac{1}{N}\sum_{n=1}^{N}\eta_n = \frac{\varepsilon N\eta}{N} = \varepsilon\eta \tag{20}$$

and the intensity diffraction efficiency of the neural network weight is given by

$$\left|w_Q\right|^2_{dependent} \approx \left(\varepsilon\eta\tau_E\right)^2 \tag{21}$$

which is independent of N. The ratio of the weights for the dependent and independent hologram cases is given by

$$\frac{\left|w_Q\right|^2_{dependent}}{\left|w_Q\right|^2_{independent}} = \left(\varepsilon N\right)^2 \tag{22}$$

The quantity ε is a measure of the degree of independence or statistical correlation of the holograms. It will also be related to the rank of the final weight matrix. If $\varepsilon=1/N$ then the holograms are independent and the intensity diffraction efficiency of the weight will decrease as $1/N^2$. This case corresponds to each weight vector adjustment being statistically uncorrelated with the previous one and can result in a full-rank weight matrix if the number of exposures is at least as large as the number of neurons in either of the layers. At the other extreme, $\varepsilon=1$ corresponds to the case in which the weight matrix is adjusted by the same outer-product throughout the exposure process, e.g. effectively only one hologram is recorded which utilizes the entire available dynamic range of the photorefractive gratings. The rank of the weight matrix is then equal to 1. The ε value for neural network models will lie between these two extremes.

We can conclude that the number of weight updates which can be stored in a photorefractive crystal during training of an optical neural network can be much larger than the number of independent holograms which can be stored. Quantitative estimates of ε will require detailed numerical simulations of optical neural networks and will vary depending on the neural network model and the problem domain.

3.2 Limitations of Conventional Holographic Weight Storage

A variety of mechanisms limit the practical holographic storage capacity of photorefractive crystals to values below the theoretical limit. Noise sources were discussed in the previous section. In this section we discuss limitations due to Bragg degeneracy and beam coupling, factors which can be alleviated using multiple-grating holography.

One of the most important impediments to holographic neural networks is crosstalk which arises from an effect known as "Bragg degeneracy". The Bragg condition states that the angle of incidence of a light beam relative to a volume grating must match one of the original writing beams in order to form an optical connection. However, even if the angular selectivity is high, crosstalk can still occur. Given a particular grating, it is possible for many light beams to satisfy the Bragg condition for that grating, in addition to the beams which originally wrote the grating. As shown in the light beam k-space diagram of Figure 6 (also known as an Ewald sphere diagram), a set of beam pairs which define the surfaces of two end-to-end cones all form the same angle with respect to the grating. All of the neuron pairs defined by the cones are connected by that grating even though it was written by only one grating pair. Therefore, a large set of beams other than the original writing beam can scatter constructively from the grating, forming erroneous reconstructions and crosstalk.

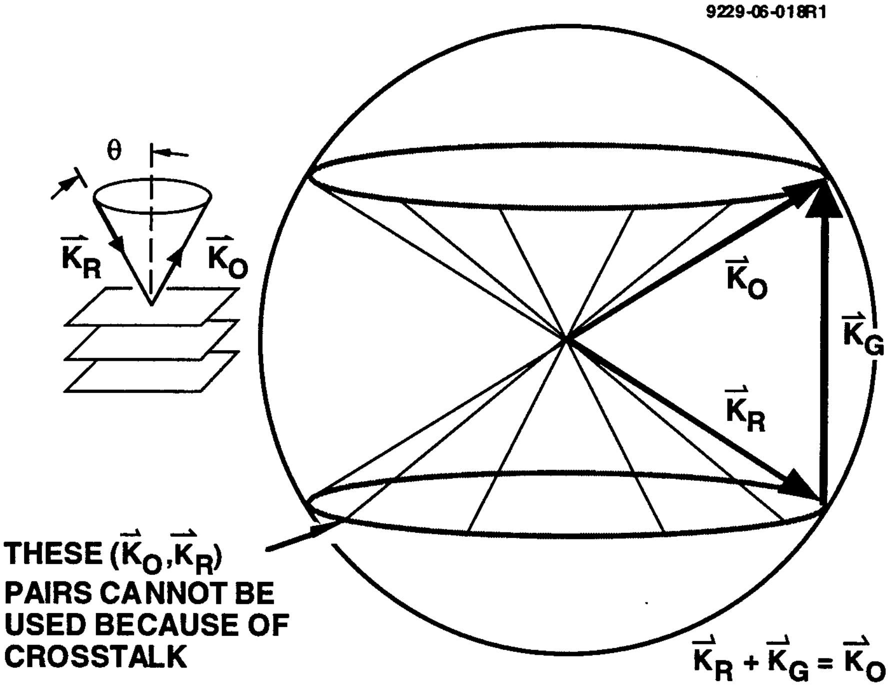
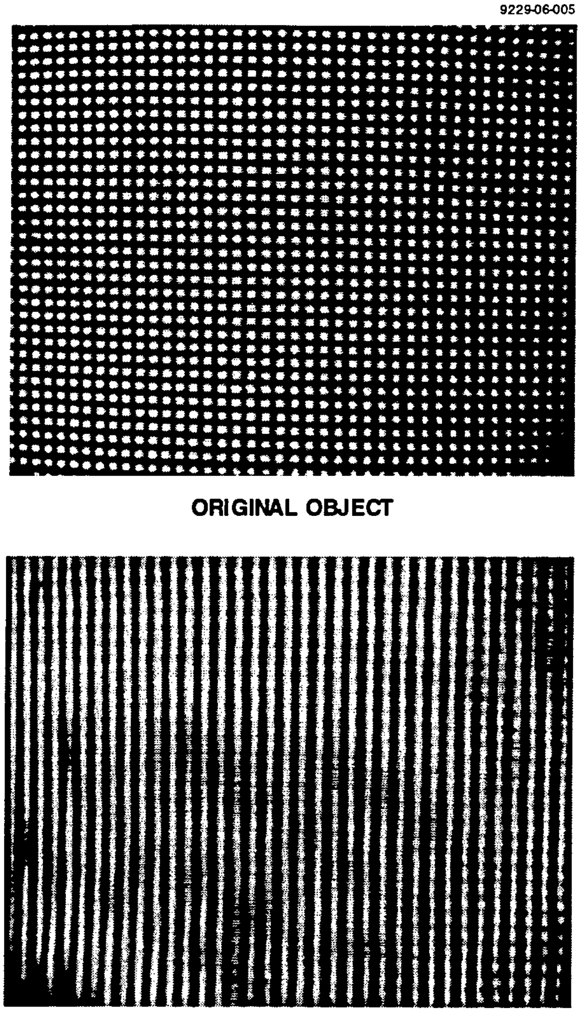

9229-06-018R1
θ
$\vec{K}_R$
$\vec{K}_O$
$\vec{K}_O$
$\vec{K}_G$
$\vec{K}_R$
THESE ($\vec{K}_O$,$\vec{K}_R$) PAIRS CANNOT BE USED BECAUSE OF CROSSTALK
$\vec{K}_R + \vec{K}_G = \vec{K}_O$

Figure 6. K-space construction of cross-talk resulting from Bragg degeneracy.

An experimental demonstration of Bragg degeneracy is shown in Figure 7. We recorded a hologram in a c-cut $BaTiO_3$ crystal using a laser diode light source with a wavelength of 830 nm. (The wavelength and crystal geometry were chosen for low two-wave mixing gain in order to eliminate beam fanning which, as will be shown later in this report, can be used to eliminate Bragg degeneracy.) The experimental configuration of Figure 4 was used. During recording the object plane consisted of a regular rectangular grid pattern and the reference plane was a uniformly filled rectangle (all pixels on). When the hologram was read out using the uniform reference, the reconstructed object was smeared in the vertical direction. No horizontal smearing can be observed. This can be easily explained by considering the k-space diagram of Figure 8. In Figure 8(a) it can be seen that the grating k-vector created by beams R1 and O1 also connects all beams above and below R1 in the R plane. These beams, such as, for example, R2, reconstruct extraneous beams such as O2 which appear above and below the original beam O1. This results in vertical smearing. In Figure 8(b) we see that horizontally displaced beams are not connected by the same grating k-vector, therefore horizontal smearing is much less. Note that Bragg degeneracy cannot be eliminated by simply phase aberrating the reference beam since the above construction would still hold for the individual plane wave components.



ORIGINAL OBJECT

HOLOGRAM OF OBJECT

Figure 7. Experimental demonstration of Bragg degeneracy using infrared hologram recorded in c-cut $BaTiO_3$. *Vertical smearing is symptomatic of Bragg degeneracy.*

Possible approaches for avoiding Bragg degeneracy are subsampling of the SLMs and spatial multiplexing of holograms. In the subsampling approach, neurons are arranged in sparse nonredundant patterns on the SLMs, and output planes are similarly sparsely sampled; thus although false reconstructions still occur, they occur at unused positions and do not contribute to the output. The special patterns can consist of so-called "fractal" lattices or other sparse patterns.[31] If the SLMs are capable of displaying $N \times N$ neurons, then this approach can implement a total of $N^{3/2}$ neurons and N^3 weights. This has the pleasing quality that the storage capacity of the crystal and the number of weights required to fully interconnect the neurons on the sampled SLMs have the same dimensional scaling. In many practical cases, however, it also has the drawback that the storage capacity may be limited by the number of neurons that can be displayed in sparse patterns on the SLM, rather than by the potentially large capacity of the crystal. As discussed above, the storage capacity of a 1 cm^3

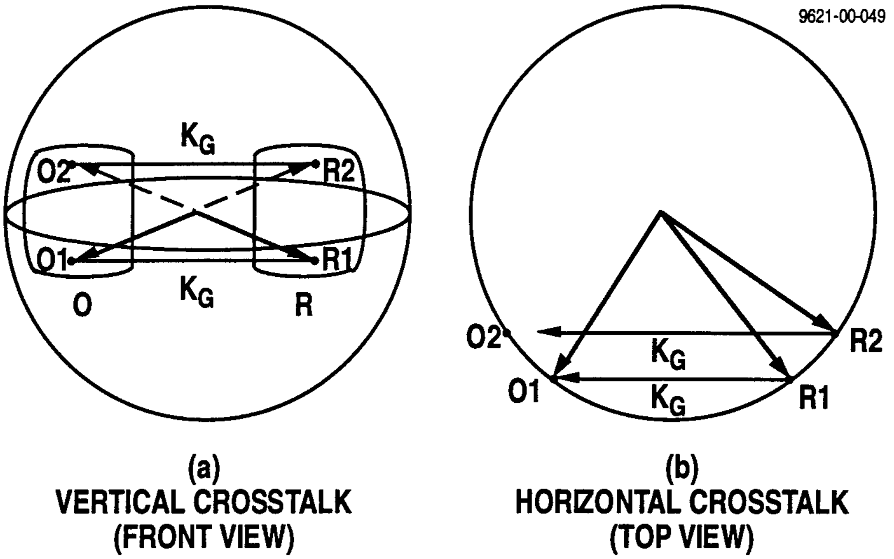


Figure 8. K-space construction showing origin of vertical smearing effect in Bragg degeneracy. *Vertically-displaced pairs of reference-object points are all connected by the same grating k-vector.*

crystal should be sufficient to store the interconnections for a N × N array of neurons where N=500, which matches the capabilities of present-day SLMs. However, because of the subsampling, only $N^{3/2}$ neurons can be implemented even though the SLM is capable of displaying N^2 neurons. Since N=500, the neuron and weight storage capacities are reduced by factors of 22 and 500, respectively, from the theoretical maximums. The light efficiency is also lowered because some of the light is diffracted to unused pixels as a result of the Bragg degeneracy.

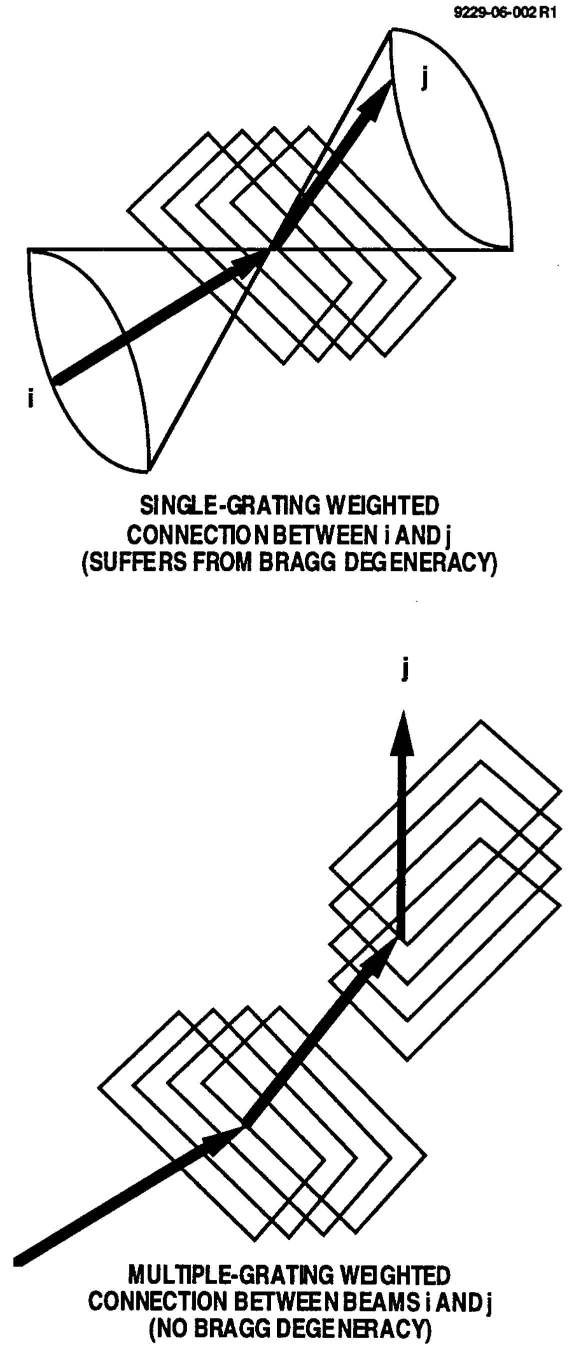
The spatial multiplexing approach avoids the Bragg degeneracy problem by physically dividing the crystal into separate volumes for each weight.[32] However, this reduces the coupling efficiency of gratings, reduces parallelism because sequential exposures must be used, and increases hardware complexity.

A second major source of distortion in holograms is energy transfer between Bragg-matched beams via two-wave mixing. This occurs because when an input beam is Bragg-matched to a grating, the reconstructed beam is automatically Bragg-matched. This beam can then itself read out the same grating and reconstruct the input beam. If the gain-length product of the grating is sufficiently high, energy transfer or beam coupling can occur periodically between the two beams, as Kogelnik showed using coupled mode analysis. Several workers have analyzed the effects of beam coupling on holographic image quality using various methods.[33,34] They found that distortion increases rapidly as the diffraction efficiency increases.

3.3 Multiple-grating Holography

We have developed a holographic recording technique called multiple grating holography (MGH) which greatly reduces the distortions due to Bragg degeneracy. The essence of the MGH idea is to use a set of angularly and spatially multiplexed gratings to store each weight rather than a single grating. By forcing a light beam to match the Bragg condition at each of a cascaded series of spatially and angularly distributed gratings (Figure 9), crosstalk due to Bragg degeneracy is greatly reduced. The k-space construction of Figure 10 shows that two gratings in series will connect only a single input/output pair of beams via an intermediary diffracted beam. All other beam triplets (input, intermediary, and output beams) will not match the Bragg conditions at both gratings because the intermediate diffracted beam will not lie on the Bragg degeneracy cone of the second grating. An undesired beam on the Bragg degeneracy cone of one grating is therefore rejected by the remaining gratings. (This allows the neurons to be arranged in arbitrary patterns on the SLM, increasing the storage capacity and light efficiency (since all pixels can be used) as well as the computational throughput. If the three beams are all in the same plane, then the minimum allowable pixel separation is greater in the vertical direction than horizontally due to the tangency of the two Bragg cones for co-planar beams.[35] However, deviations from co-planarity as small as the Bragg width are sufficient to eliminate the tangency of the Bragg cones. Our experiments show that pixel separations typical of SLMs are sufficient to eliminate crosstalk in both directions.

Two techniques for generating such multiple-grating connection weights in photorefractive materials have been investigated: self- and mutually-pumped phase conjugate mirrors (PCM)[36-39] and forward-scattering beam fanning.[40] Although effective in reducing crosstalk due to Bragg degeneracy, the PCM method results in very strong inter-hologram crosstalk due to the hologram-sharing effect.[41] This led to the beam fanning method which also eliminates Bragg degeneracy but does not suffer from hologram sharing.



SINGLE-GRATING WEIGHTED CONNECTION BETWEEN i AND j (SUFFERS FROM BRAGG DEGENERACY)

MULTIPLE-GRATING WEIGHTED CONNECTION BETWEEN BEAMS i AND j (NO BRAGG DEGENERACY)

Figure 9. Optical connections made by scattering from multiple cascaded gratings reduce cross-talk due to Bragg degeneracy and allow full utilization of input and output planes.

"Beam fanning" is a well-known effect in high gain photorefractive crystals in which an input beam is initially scattered by small inhomogeneities in the crystal, resulting in low amplitude scattered optical noise.[42] The noise beams then interfere with the original input beam and write gratings.

Scattering of the input beam by these gratings selectively amplifies some of the noise beams by the process of energy transfer in photorefractive two-wave mixing. The amplified beams then write new gratings and the process cascades. Which beams are amplified most is determined by the electrooptic tensor of the crystal and the orientation of the input beam. The net effect is that the input beam literally fans in the crystal as it writes a set of spatially and angularly distributed gratings.

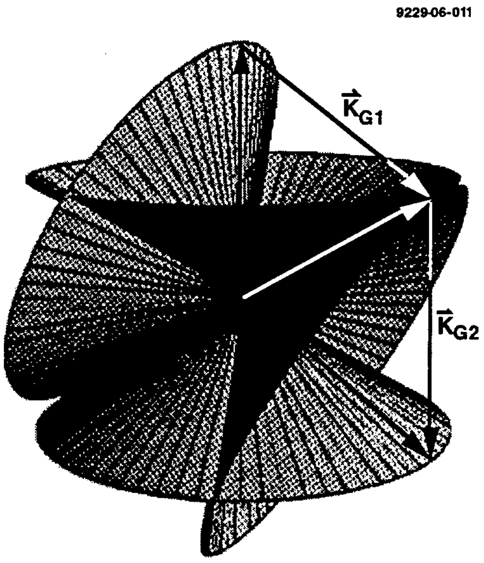


Figure 10. K-space construction for satisfaction of Bragg conditions at two gratings simultaneously. *Only one triplet of light beams satisfy the Bragg conditions of both gratings.*

As shown in Figure 11, this fanned light can be used as a reference beam when it is interfered with a second, unfanned object beam to form a holographic connection which suffers neither from Bragg degeneracy (because multiple cascaded gratings store each connection) nor from hologram sharing (because the conjugate beam is not used for readout). An unfanned object beam is used because object beam fanning would degrade the quality of the reconstructed object image. Fanning can be controlled so that the reference beam fans and the object beam doesn't by adjusting the orientation of the beams relative to the crystal. The fanning process generates high gain-length product "fanning gratings" which divide each reference beam into a set of beams at different orientations and locations. During recording the object beams form a set of "signal gratings" in which the connection weights are stored. Both the signal and fanning gratings are angularly and spatially multiplexed. Upon readout each reference beam must match the Bragg condition at a multitude of fanning gratings, which breaks the Bragg degeneracy. In addition, the beamlets reconstructed by the signal gratings are all in phase and sum coherently, hence the aggregate diffraction efficiency can be high even though the diffraction efficiency of any individual signal grating is small. In addition, the low gain-length product of the individual signal gratings greatly reduces distortions due to beam coupling.

Recordings of holograms in a fanning crystal of $BaTiO_3$ do not exhibit observable Bragg degeneracy, hologram-sharing crosstalk, or distortions due to beam coupling. An example of holographic recording using an arbitrary 2-D gray-scale reference image is shown in Figure 12. The object and reference both consisted of combinations of Pentagon and woman images displayed on a LCTV with 30,000 gray-scale pixels (90,000 pixels if the individual red, green, and blue pixels of the color LCTV are counted). The original object (showing the system's optical quality) and reconstructed hologram are shown in Figures 12(b) and (c). Note that only 50% of the original reference was used in reading out the hologram. The reconstructed image is virtually identical to the original. Upon

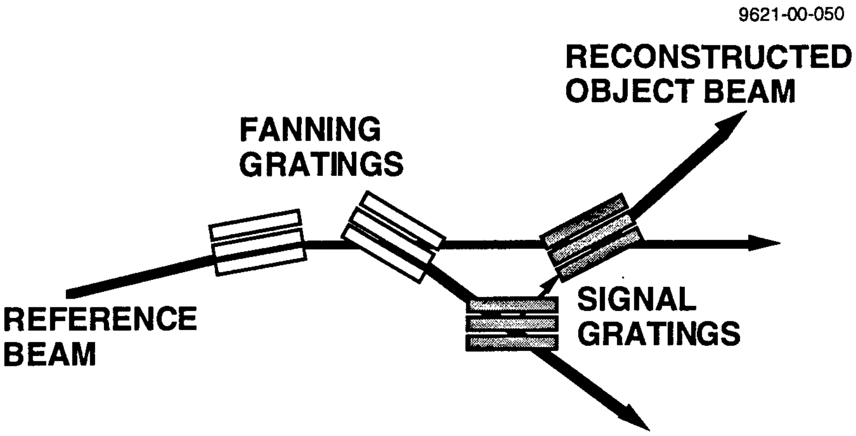


Figure 11. Recording of connection using fanned reference beam.

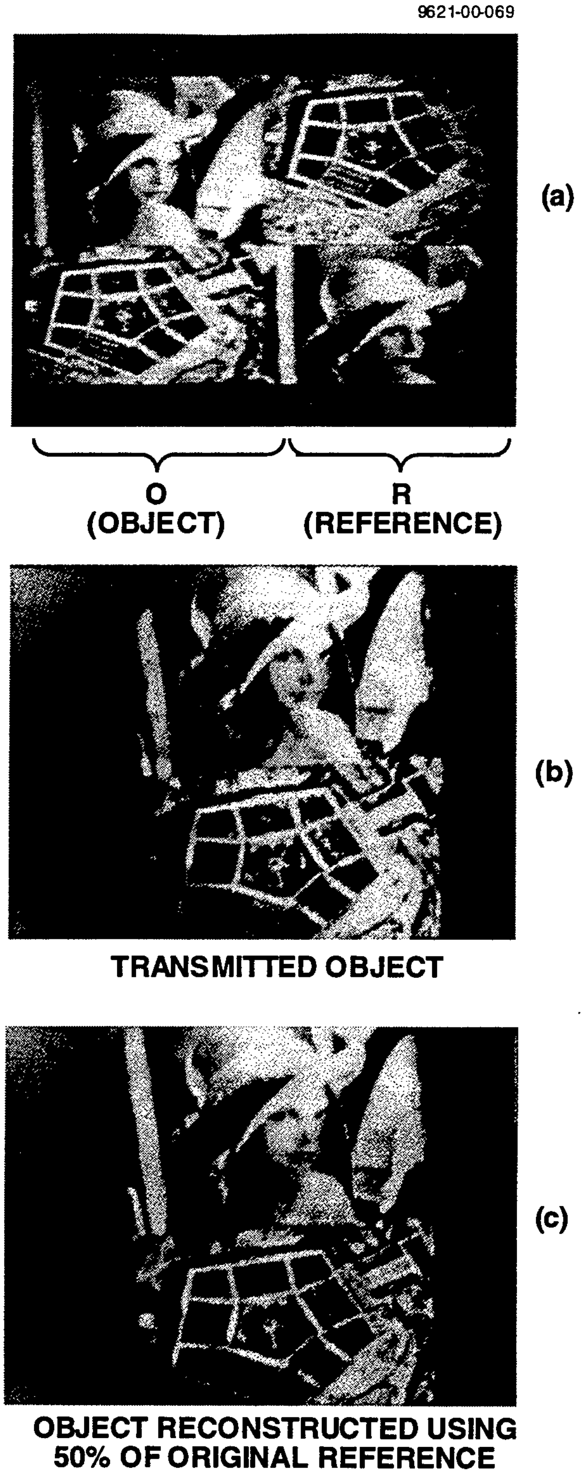
magnification of the reconstructed hologram, each of the 45,000 LCTV pixels in the original object was clearly visible.

We also performed experiments to measure the holographic fan-out. In one experiment a uniform (all pixels on) reference was used to record the hologram. During readout an opaque screen with an adjustable small round aperture was translated in front of the reference plane, allowing us to measure the weight vector connected to that portion of the reference. In this experiment the aperture area was less than 1% that of the reference, corresponding to a fanout of over 100. (Measurements of larger fanouts were limited by the sensitivity of our CCD camera.) The entire object was reconstructed when read out with 1% of the reference, demonstrating global connectivity.

We also measured the time required to write holograms to saturation using fanned reference beams. The results are shown in Figure 13 for a crystal of $BaTiO_3$ which was cut at 45° to the c-axis. This crystallographic orientation maximizes the effective electrooptic coefficient.[43] We used two slotted wheels on a common rotating shaft to periodically switch off the object beam and unblock a photodiode which measured the diffracted light. In this way we could measure the buildup in diffraction efficiency as the hologram was being written. (This accounts for the pulsed nature of the oscillioscope trace in Figure 13.) In this case the wavelength was in the green (514 nm) and the total optical power incident on the crystal was 17.5 mw. The time required to write to saturation was 25 msec. Since weights are typically adjusted by small amounts during the learning phase of neural networks, all the weights connecting two neuron layers can be incrementally adjusted in less than 1 msec. This would correspond to learning rates of greater than 10^{11} connection updates per sec if each layer contained 10^4 neurons. Since the photorefractive time constant is roughly inversely proportional to the optical power, the update rate can be further increased by raising the laser power.

A demonstration of recording superimposed holograms using fanned reference beams is illustrated in Figure 14. In this case the objects were rotated versions of a gray-scale woman image and the references were orthogonal grid patterns. As shown in Figures 14(b) and (c), each of the superimposed holograms could be read out with very little crosstalk between holograms and with approximately equal diffraction efficiencies.

Improvements in packaging size and cost would be obtained if a laser diode light source could be used. Currently available $BaTiO_3$ is less sensitive at the infrared laser diode wavelengths and the photorefractive gain is less. Nevertheless, we were successful in demonstrating multiple-grating holography in 45°-cut $BaTiO_3$ using a laser diode light source operating at 830 nm. This crystallographic orientation maximizes the effective electrooptic coefficient, which is required for maximum beam fanning. We used a 100 mw single mode laser diode. The total optical power at the crystal was 31 mw. As shown in Figure 15(a-c), excellent holographic image quality without Bragg smearing can be obtained in the near-infrared in $BaTiO_3$ using multiple-grating holography. Low signal-to-noise ratio images could be recorded with exposure times as short as 67 msec, as shown in Figure 15(d). Nevertheless, due to the much reduced sensitivity of $BaTiO_3$ at 830 nm, the time constant for hologram writing was much longer than at 514 nm. We have measured a time to saturation of 10 sec for an incident power of 45 mw at 830 nm. Thus more work is needed to improve the infrared sensitivity of the recording medium for use with laser diodes. Alternatively, compact solid state lasers emitting in the green could be used.[44]



TRANSMITTED OBJECT

OBJECT RECONSTRUCTED USING 50% OF ORIGINAL REFERENCE

Figure 12. Experimental demonstration of recorded image quality using fanned reference beam and 2-D non-subsampled reference. *(a) Object and reference used for recording, (b) Original object for comparison, (c) Hologram of object reconstructed using 50% of original gray-scale reference.*

9621-00-072

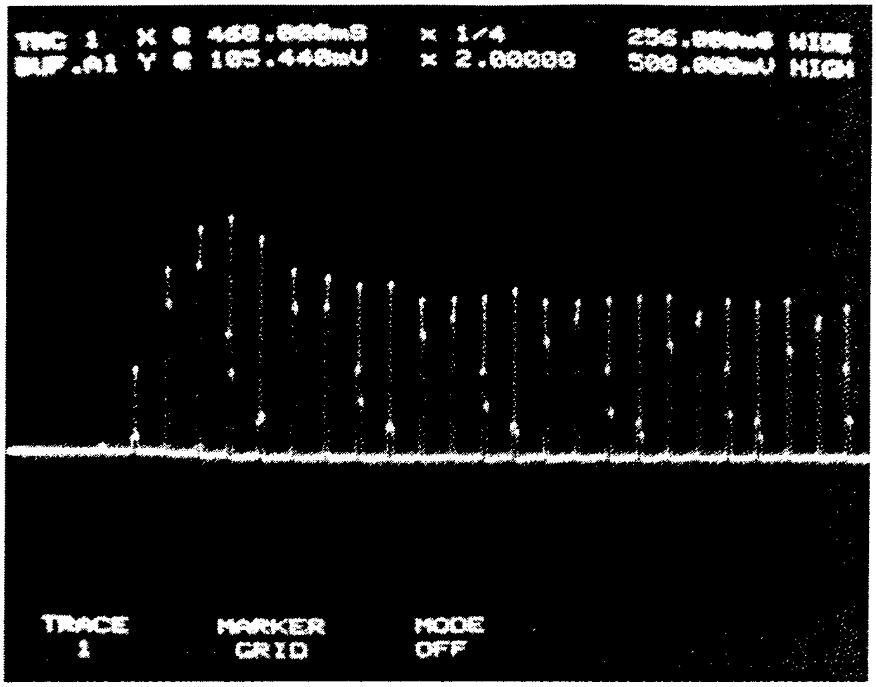
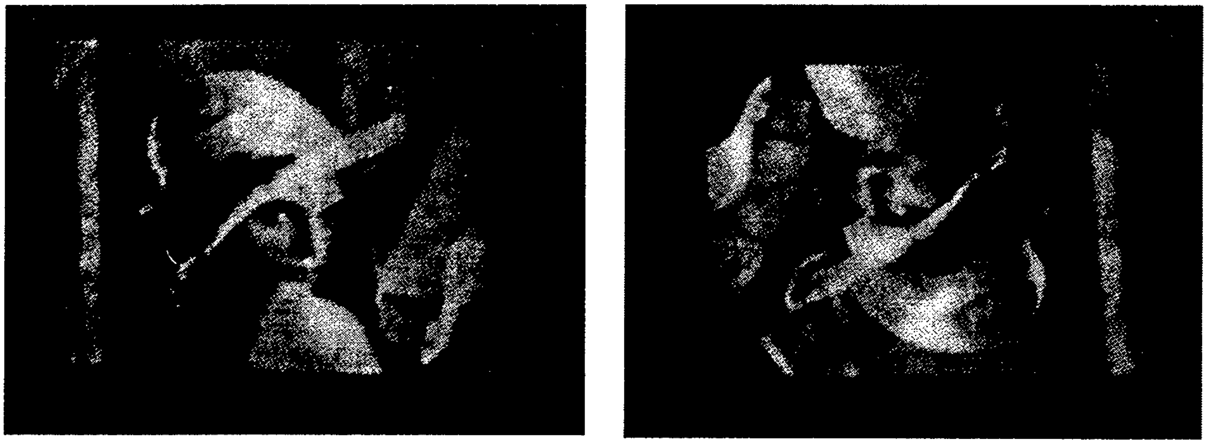


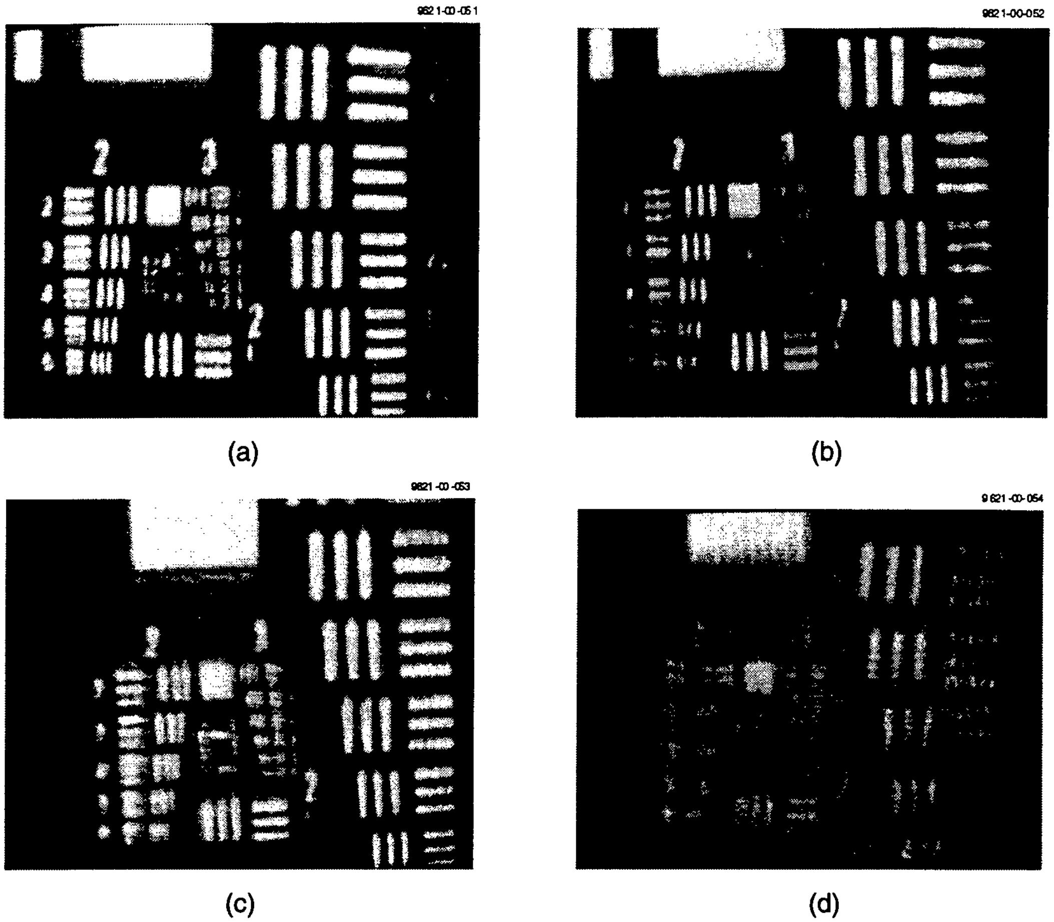
Figure 13. Holographic time response of 45°-cut $BaTiO_3$ for writing wavelength of 514 nm.

9621-00-073



(a) **(b)**

Figure 14. Demonstration of superimposed holograms using fanned reference beams.



(a)

(b)

(c)

(d)

Figure 15. Demonstration of fanned-reference holography in $BaTiO_3$ using laser diode light source with a wavelength equal to 830 nm. *(a) Original object as seen through the crystal, (b) Hologram of object after 20 sec exposure, (c) Hologram after 0.5 sec exposure, (d) Hologram after 0.067 sec exposure.*

4. LASER-DIODE-BASED HOLOGRAPHIC NEUROCOMPUTER

Neurocomputers should be characterized by, first, large numbers of neurons and weights; second, software mapping of a variety of neural network algorithms; third, co-processor-type interfacing to a host computer; and fourth, hardware simplicity for low cost and practical packaging. In this section we describe an experimental optical neurocomputer which serves as a testbed for some ideas which we hope contribute to achieving these goals. A variety of neural networks, including Perceptron and backpropagation networks, have been successfully demonstrated on this system.

4.1 Experimental Set-up and Packaging Concepts

A diagram of the SPONN (stimulated photorefractive optical neural network) neurocomputer is shown in Figure 16. It is a hybrid optical/electronic system: although the weighted interconnections of the neural network are implemented holographically, thus taking advantage of the parallelism and three-dimensional connectivity of optics, the neuron nonlinearities and data input/output are implemented digitally. The reference and object input planes containing the optical neurons are displayed side by side on the spatial light modulator (SLM) using two Variable Frame Grabber (VFG) image processing cards from Imaging Technology in the host computer. One VFG board is dedicated to displaying neuron outputs on the SLM while the other is used to grab video from the CCD detector which contains the summed inputs to the neurons. Nonlinear point operations such as the neuron sigmoidal transform are implemented in the host computer (a 66 MHz 486 PC). The SLM is a 1" × 1" VGA-resolution (640 × 480 pixels) active-matrix liquid crystal display panel from Kopin Corp. The input planes modulate the reference and object light beams which are directed into the photorefractive crystal where holograms interconnecting the two input planes are written. The light source is a 200 mw single-mode laser diode from SpectroDiode Labs operating at 837 nm. The photorefractive crystal is rhodium-doped $BaTiO_3$ grown at Hughes Research Labs which has been shown to have enhanced sensitivity at laser-diode wavelengths.[45] This generation of the SPONN system uses off-the-shelf optical components and was not optimized for minimum volume. It occupies an optical breadboard measuring 2' × 2'.

We have done detailed optical designs which show that the same components could be mounted on a double-sided breadboard measuring 1' × 1', as shown in Figure 17. In this concept, a miniaturized floating breadboard will be utilized to construct the neural-net. Figure 17 illustrates this design approach schematically. To keep the system compact, the optical path is folded so that both the laser source-beam collimating assembly and the image reconstruction apparatus are mounted on the backside of the breadboard. The processing unit composed of the SLM, the object and reference paths, and the nonlinear crystal are all mounted on the opposite side of the breadboard. The breadboard itself is hung and floated in a 3-D cubic frame. The floating mechanism is established by holding the breadboard with two-way shock absorber posts at the four corners. This arrangement would isolate the breadboard unit from the external interference so that stability can be achieved. Based on our

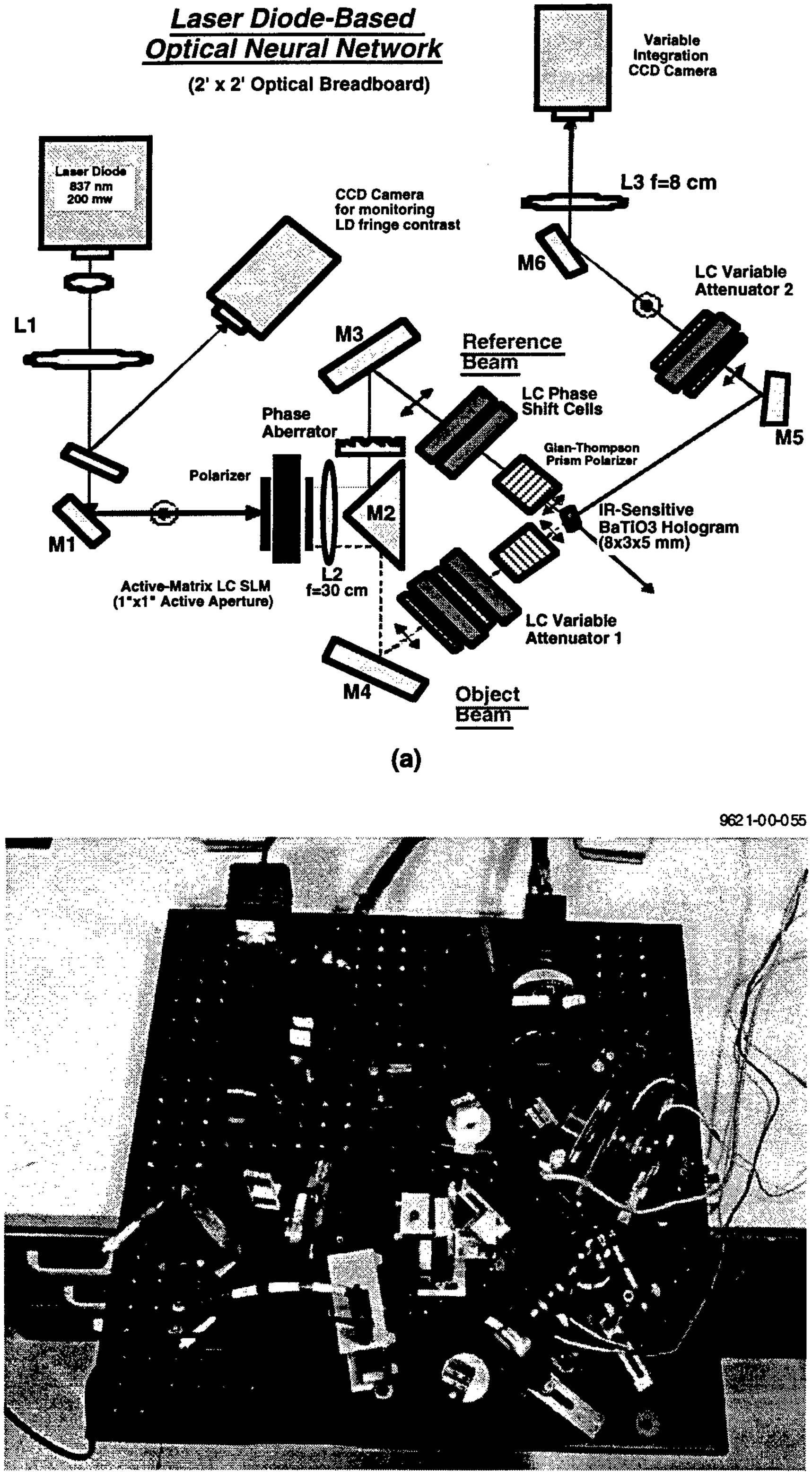
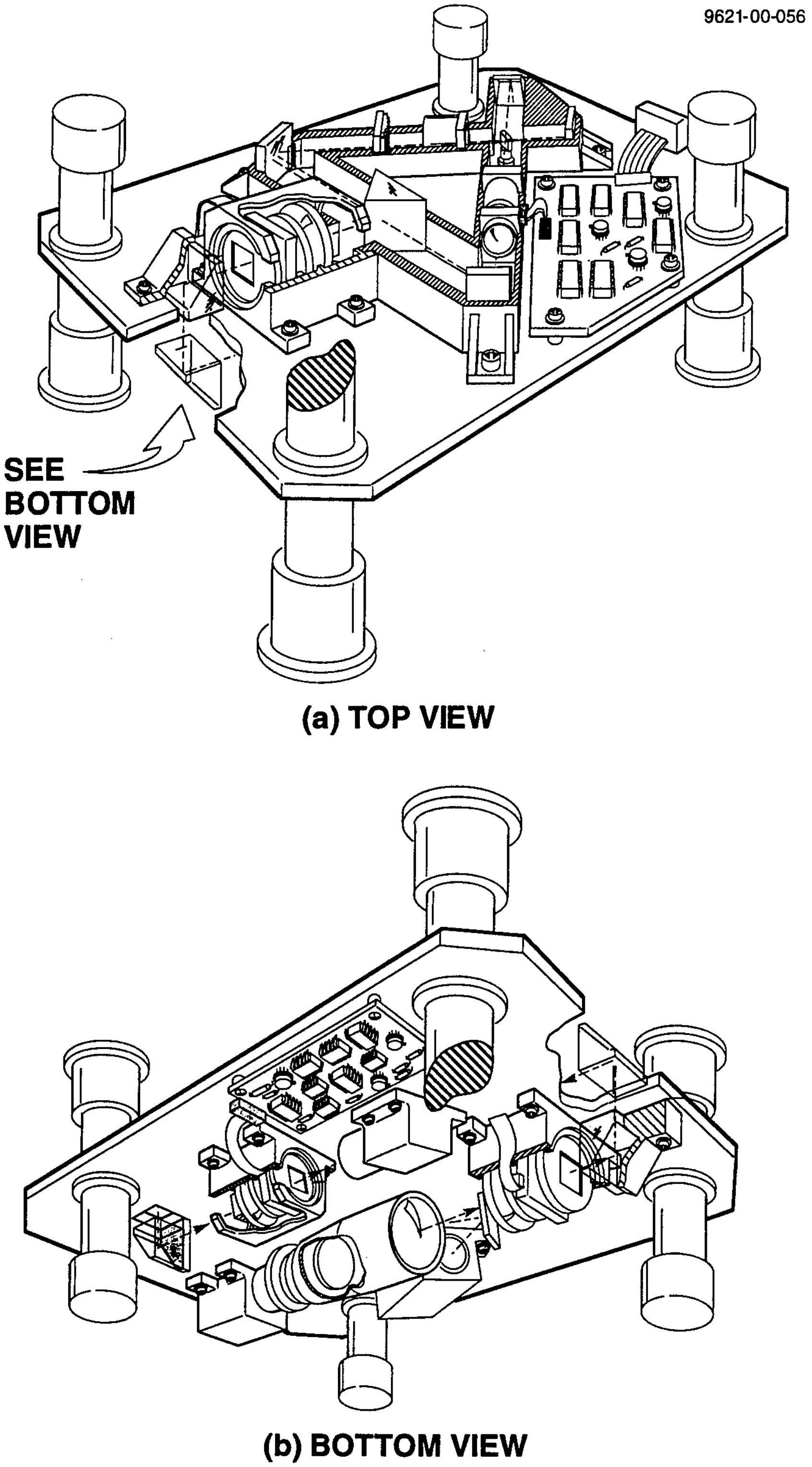

Laser Diode-Based
Optical Neural Network
(2' x 2' Optical Breadboard)
Variable Integration CCD Camera
Laser Diode 837 nm 200 mw
L3 f=8 cm
CCD Camera for monitoring LD fringe contrast
M6
LC Variable Attenuator 2
L1
M3
Reference Beam
LC Phase Shift Cells
Phase Aberrator
M5
Glan-Thompson Prism Polarizer
Polarizer
M2
IR-Sensitive BaTiO3 Hologram (8x3x5 mm)
M1
Active-Matrix LC SLM (1"x1" Active Aperture)
L2 f=30 cm
LC Variable Attenuator 1
M4
Object Beam
(a)
9621-00-055
(b)

Figure 16. (a) Layout of laser-diode-based optical neurocomputer, (b) Photograph of laser diode based optical neurcomputer on 2'×2' optical breadboard.



(a) TOP VIEW

(b) BOTTOM VIEW

Figure 17. Packaging concept for optical neurocomputer using off-the-shelf components. *The neurocomputer occupies a 1'x1' two-sided optical breadboard mounted on shock absorbers. (a) Top view, (b) Bottom view.*

experience in opto-mechanical engineering, the stability of mounting and adjusting hardware will be the key to the success of this approach. It is recommended that special mounting fixtures, such as the gimbal type mounts, with fine tuning and lockable features should be considered. This packaging concept is aimed at building a compact neural network at a brassboard level allowing limited modification or alignment work to be accommodated. It is based on the principle of isolating the system from the environment. Use of a smaller SLM, such as the display panels now available for helmet-mounted displays, would allow shorter focal length optics and further reduce the size of the system.

The operation of the neurocomputer consists of separate hologram writing and readout phases. In the hologram writing phase, the reference beam interferes with the object beam in the crystal, forming the cascaded-grating connection hologram. In the readout phase, the reference beam reconstructs the object beam, which is detected by a variable frame time CCD camera. The CCD output is converted into digital form, allowing the neuron sigmoidal nonlinearity to be implemented digitally in the host computer. Dual image processing cards in the PC host are used to process the input and output planes. By manipulating the input plane using one of the image processing cards, various single and multiple-layer neural network models can be implemented on the same optical hardware. Liquid crystal cells are used to vary the amplitude and relative phase of the reference and object beams. Phase shifting is used in the balanced coherent detection method discussed below for representing bipolar neurons and weights.

4.2 Bipolar Weight Representation

Most neural network models require weights and neuron outputs to assume both positive and negative values. Such bipolar quantities are necessary even if the neuron response function saturates at 1 and 0 in order to be able to both rectify wrong responses and reinforce correct responses without saturating the outputs. In addition, some neural networks, such as backpropagation, require the detection of a bipolar error signal. Therefore means must be provided in an ONN for bipolar inputs and outputs. Holographic ONNs can use coherent or incoherent methods for representing bipolar inputs and outputs.

In coherent approaches direct phase modulation of light is used to shift the phase of gratings during writing. Upon reading out the hologram, the phase of diffracted light beams is measured by mixing with a reference beam and using interferometric detection. Interferometric detection has potential benefits in terms of increased dynamic range, but it also has many practical difficulties. The phase shifting of the input must be done with great uniformity across the entire SLM input. (Although this can also be accomplished using Stokes' principle[46] or phase modulating devices and exposing the positive and negative components separately.) More problematic is the interferometic detection at the output detector array. It is very difficult to maintain phase uniformity across the entire output detector array. Increased vibration sensitivity is also a problem. In addition, many presently available SLMs have an amplitude-dependent phase response which makes independent control of phase and amplitude impossible without using an additional compensating phase-only SLM, which would greatly increase alignment difficulties and system complexity.

In order to avoid these problems, in our first SPONN system we represented bipolar inputs, weights, and outputs incoherently using spatial multiplexing. A bipolar input y_j (which could be an input from neuron j or an error signal) is divided into two nonnegative quantities y_j^+ and y_j^- where

$$y_j^+=y_j \text{ and } y_j^-=0 \quad \text{if } y_j>=0$$

$$y_j^+=0 \text{ and } y_j^-=|y_j| \quad \text{if } y_j<0 \tag{23}$$

These operations are performed electronically in the host computer. y_j^+ and y_j^- are then written to two spatially-separated SLM pixels in the reference half of the optical input plane. For the writing or weight adjustment phase of a neural network algorithm, two similar nonnegative quantities are written to two SLM pixels in the object section of the input plane, representing a bipolar error signal in neuron i. The crystal is then exposed with the object and reference beams, forming four intermediate weights w_{ij}^{++}, w_{ij}^{+-}, w_{ij}^{-+}, and w_{ij}^{--} which encode a bipolar effective weight connecting neurons i and j. w_{ij}^{++} increases if both object and reference are positive, w_{ij}^{+-} increases if object is positive and reference is negative, and so on for w_{ij}^{-+} and w_{ij}^{--}.

The bipolar algorithm for the readout phase is illustrated in Figure 18. The effective weight is decoded by reading out twice, once with the input and again with its negative and then subtracting the two outputs. Input y_j is again split into positive and negative parts which read out the weights in the hologram. Square-law detection of the diffracted light beams is performed at two CCD pixels, forming the intermediate terms

$$\begin{aligned} d^+ &= \left| \sum_j w_{ij}^{++} y_j^+ + \sum_j w_{ij}^{+-} y_j^- \right|^2 \\ d^- &= \left| \sum_j w_{ij}^{--} y_j^- + \sum_j w_{ij}^{-+} y_j^+ \right|^2 \end{aligned} \tag{24}$$

(In our optical system many CCD pixels are actually used for each output so that spatial averaging as well as temporal averaging can be used to reduce noise.) The square-root of each of these CCD outputs is formed before subtracting them. The final output after subtracting the two readouts is

$$\begin{aligned} out_1 &= \left(\sqrt{d^+} - \sqrt{\mathrm{d}-}\right)_{+INPUT} - \left(\sqrt{\mathrm{d}^+} - \sqrt{\mathrm{d}}\right)_{-INPUT} \\ &\propto \sum_j \left(w_{ij}^{++} + w_{ij}^{--} - w_{ij}^{+-} - w_{ij}^{-+}\right)\left(y_j^+ - y_j^-\right) \\ &\propto \sum_j w_{ij} y_j \end{aligned} \tag{25}$$

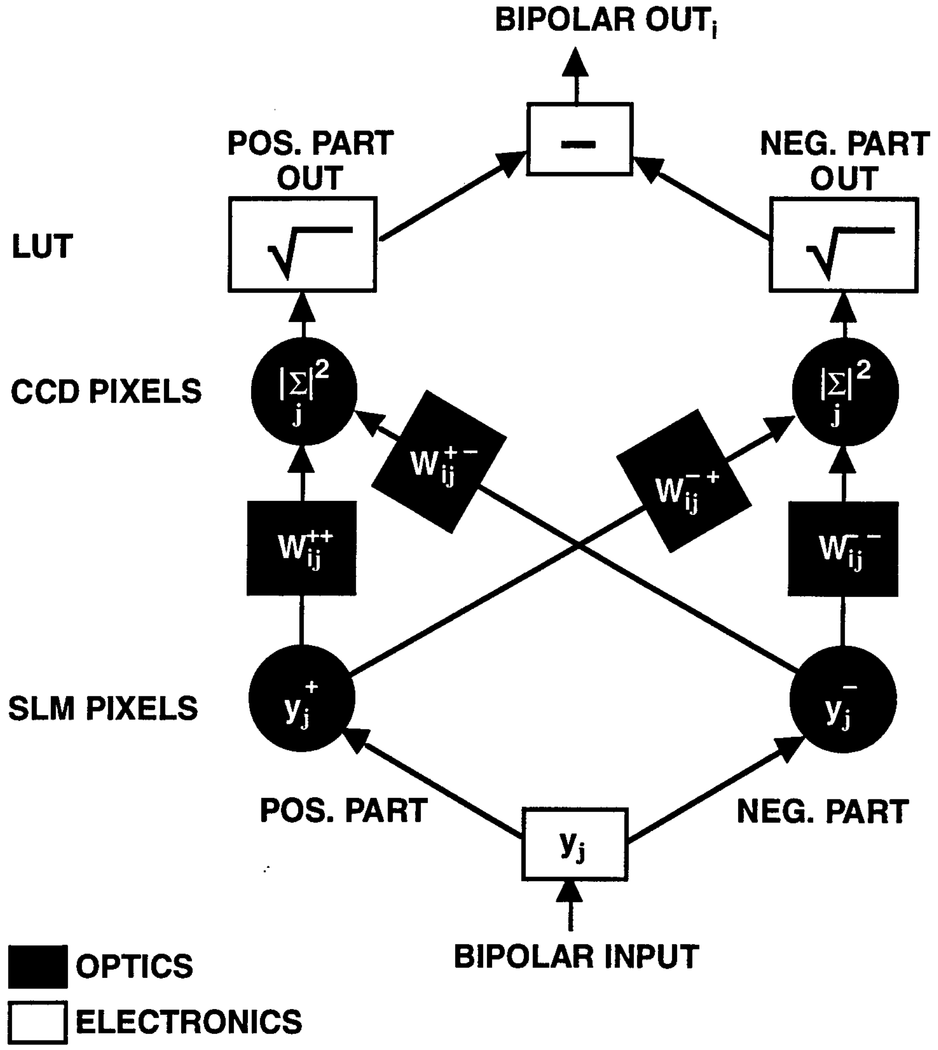


Figure 18. Algorithm for bipolar representation of weights and neuron values.

which represents a true bipolar output with bipolar weights and inputs. The electronic portions of the algorithm are not bottlenecks. The square-roots are performed at video rates on the entire video frame using a lookup table. Note that this method relies on the fact that all optical terms have the same phase, namely that of the object beam at the respective CCD pixel.

We also place y_j^+ and y_j^- on a bias Δ so that the actual optical inputs for each of the two pixels which comprise a neuron are $\Delta+y_j^+$ and $\Delta-y_j^-$. This ensures that the overall intensity level in the photorefractive crystal, and hence the photorefractive time constant, is independent of the input values. The addition of a bias does not change the above results.

In our earlier SPONN system described above we represented bipolar neurons and weights by using separate pixels for positive and negative neuron values and taking the difference to form the final output. We did not use the "dual-rail" representation scheme in the second SPONN system because we have found that bias terms build up in the positive and negative rails. In theory, these terms cancel in the final result. In practice, however, the dual-rail representation increases the dynamic range requirements of the system. Small output values must often be calculated by subtracting two almost equal large numbers, resulting in poor accuracy. We have observed this effect both experimentally and using numerical simulations. In order to eliminate this effect, we have switched to a balanced coherent detection technique for representing bipolar neurons and weights.

The optical signal read out from the hologram is superimposed on the CCD detector with a constant optical bias which is allowed to pass through the hologram on the object beam from the SLM. Let b represent the bias amplitude, a the signal amplitude, and φ the phase shift between b and a induced by the LC cell in the reference beam. The CCD output as a function of φ is then

$$I(\varphi) = \left|b + ae^{i\varphi}\right|^2 = b^2 + |a|^2 + 2b\,\mathrm{Re}\left(\mathrm{ae}^{\mathrm{i}\varphi}\right) \tag{26}$$

where we have defined the phase of b as zero. Note that the output contains both constant and signal-dependent bias terms. In the balanced coherent detection method, the hologram is read out twice with φ equal to 0 and π. The outputs are stored and subtracted to give the final output:

$$V = I(\varphi = \pi) - \mathrm{I}(\varphi = 0) = 4b\,\mathrm{Re(a)} \tag{27}$$

Note that V is bipolar in a and is bias-free. In addition, positive and negative weights are stored by recording phase-shifted gratings in the photorefractive crystal using the LC phase-shifter cell in the reference beam. (We were also able to modify the phase by modulating the laser diode current, but we found this resulted in greater sensitivity to vibrations.) This eliminates the weight bias buildup problem in the dual-rail approach. Experimental results for writing and reading positive and negative weights in a $BaTiO_3$ crystal using the balanced coherent detection method are shown in Figure 19.

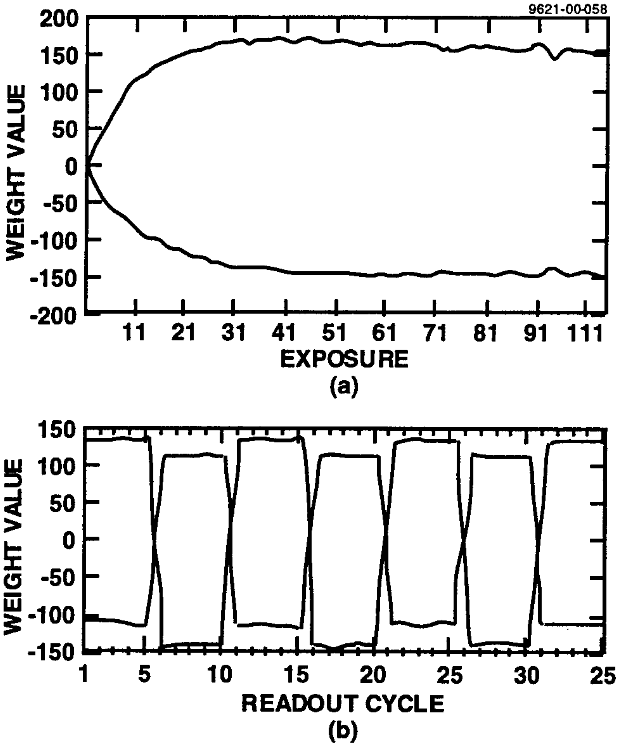


Figure 19. (a) Balanced coherent-detection readout of positive and negative weight values in a $BaTiO_3$ crystal as they are being written. (b) Readout (using balanced coherent-detection) of positive and negative weights stored in a $BaTiO_3$ crystal. The phase of the readout beam was periodically shifted by π.

5. HOLOGRAPHIC NEUROCOMPUTER EXPERIMENTAL RESULTS

The current performance is approximately 2 × 10^7 weights processed at 10^8 weights per second. The processing rate can be increased most easily by increasing the electronic I/O bandwidth. In the following sections we describe results for specific neural network models which were all programmed on the same optical neurocomputer described above without any hardware adjustments.

5.1 Perceptron

The Perceptron was one of the first neural networks to be developed.[47] In the most commonly implemented form it consists of a single layer of weights connecting a field of input neurons with a single output neuron. (Some of the original Perceptron networks contained a "preprocessing" layer with fixed weights which were not adjusted during learning.) A single output neuron can dichotomize or separate the vector space of input patterns into two classes. The weight values together with the threshold value of the output neuron determine a separating hyperplane for pattern vectors. Therefore it can only dichotomize classes which are linearly separable. The Perceptron is appealing as a first test of neural hardware because of its simplicity and the fact that a learning rule with guaranteed convergence is known (provided a solution can be represented in the first place). Its main weakness is that as a single-layer network it is limited to linearly-separable solutions and therefore cannot solve many problems of practical interest.

$$y = h\left(\sum_j w_j x_j - \theta\right) \tag{28}$$

where w is the weight vector connecting the input neurons x with the output neuron, θ is the output neuron threshold value, and h(z) is a hard-threshold response function with outputs 1 or -1 for $z>0$ and $z<=0$, respectively. Input patterns are binary and normally assume values of 1 or 0. θ can be learned by setting one of the input neurons to 1, although in our experiments we set $\theta=0$. The learning rule is simple and can be expressed in terms of the weight update vector as

$$\Delta w_j = -\eta(y - D_c)x_{j,c} \tag{29}$$

where D_c is the desired output for exemplar c. If the output is correct, no change is made. If $y=1$ but $D_c=-1$, a change proportional to the negative of the exemplar is made. Finally, if $y=-1$ but $D_c=1$, a change proportional to the exemplar is made. A slight modification of this algorithm was made in the optical implementation. A "forbidden zone" centered on zero was defined for neural outputs before thresholding: if the value of $\Sigma w_j x_j - \theta$ was within this zone then a correction to the weights was made even if the thresholded network output was correct. This made the system more robust by penalizing small weight values.

In our implementations of the Perceptron neural network, the R plane contained the pattern in the input neuron layer L_0 and the O plane contained the output neuron layer L_1. We used both single and multiple output neurons. The latter case is equivalent to many Perceptrons operating in parallel on the same input patterns but with different classification goals. In all of the experimental examples described here the problem to be solved was the tranformation of a set of random binary exemplars (2-D random patterns with pixel values of 1 and 0) into another set of random binary patterns.

The first Perceptron experiment had a single output neuron, the goal was to dichotomize a set of random patterns into two classes. The results of this experiment is shown in Figure 20(a) which is a plot of total error during learning versus epoch number. In this case the optical neural network learned to dichotomize 96 random patterns after 29 epochs. Each pattern consisted of 1920 pixels (60 × 32). Increasing the number of pixels tended to reduce the number of patterns that could be learned although with 7680 pixels (120 × 64) the system could still learn 42 patterns. In order to test the noise level in the system we attempted to classify two patterns into two classes. The patterns were identical except for a prescribed number of differing pixels. We continued to reduce the number of differing pixels until the optical neural network could no longer distinguish them. The system could separate patterns containing as few as 0.5% differing pixels. The spatially-multiplexed incoherent method for representing bipolar weights (see Figure 18) was used in Figure 20(a). Results for Perceptron learning using the balanced coherent detection method for representing bipolar weights are shown in Figure 20(b).

We were also able to implement multiple-output-neuron Perceptron networks. The networks learned to perform a one-to-one transformation of a given set of random binary patterns (values 1 and 0) into another set of random patterns (values 1 and -1). The results of one experiment are shown in Figure 21(a) in which the input and output layers contained 1740 and 870 neurons, respectively. This network had a total of 1.5 million weights. We then scaled the network up to 10,260 neurons and 2 × 10^7 weights. The learning curve for this larger network is also shown in Figure 21(b). The processing rate was 2 × 10^7 connections updated per second during learning. This rate was limited by the PC host computer because we transfered neuron values back and forth between the image processor cards and host memory using the PC bus.

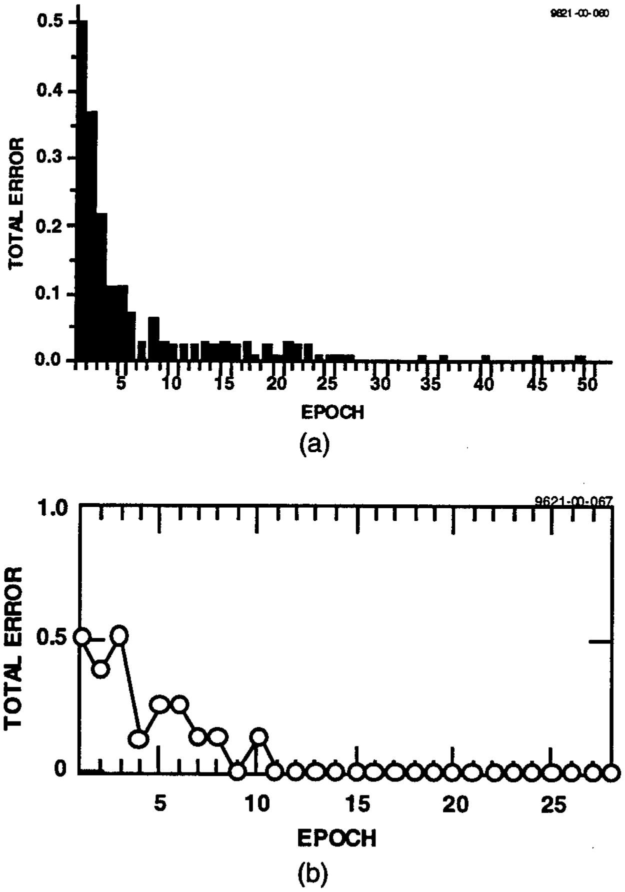


Figure 20. (a) Optical Perceptron learning of 96 exemplar patterns with 1920 input neurons and incoherent detection method. (b) Optical Perceptron learning using balanced coherent detection method.

We also used this type of network in a handwritten digit recognition application. Exemplars were extracted from a database of handwritten digits supplied by the U.S. Post Office. Examples of the digits are shown in Figure 22. The network, which consisted of 625 input neurons and 10 output neurons, was trained to label inputs as one of the ten digits. The total error during learning of 80 exemplars is also shown in Figure 23. We did not expect good performance because this network contains a single layer of weights and the character recognition problem is not linearly separable. Even so, the network achieved an 92% correct recognition rate on the training set and a 75% correct recognition rate on 40 test digits it had not seen before. Although less than required for practical applications, this result demonstrated generalization since the expected untrained recognition rate is 10%.

5.2 Backprogation

Perceptrons are interesting as a first test of neural hardware. However, the usefulness of single-layer networks is limited to separable problems. The addition of a hidden layer greatly increases the power of a neural network. As mentioned previously, it has been shown that a network with a single hidden layer (input, hidden, and output neuron layers with two layers of weights) can approximate any function provided a sufficient (but finite) number of neurons is available. One of the most popular learning algorithms in terms of applications is backpropagation.[48,49] Backpropagation is a learning algorithm based on steepest descent of an error surface defined by

$$E=\sum_{E}\sum_{i}\left(y\frac{(n)}{i,c}-D_{i,c}\right)^{2} \qquad (30)$$

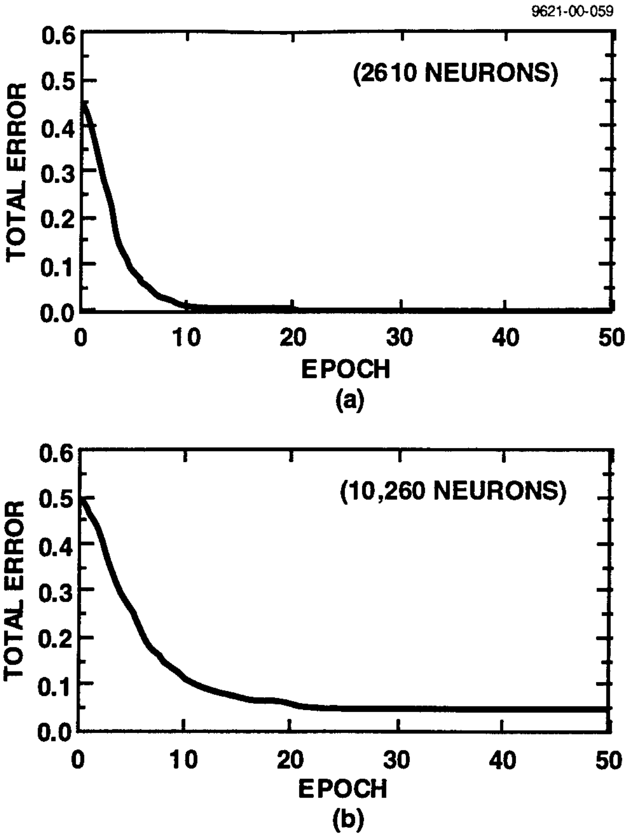


Figure 21. Multi-output-neuron single-layer optical Perceptron learning of 4 random input/output associations. *(a) 1740 input neurons, 870 output neurons, and 1.5×10⁶ weights, (b) 6840 input neurons, 3420 output neurons, and 2.3×10⁷ weights.*

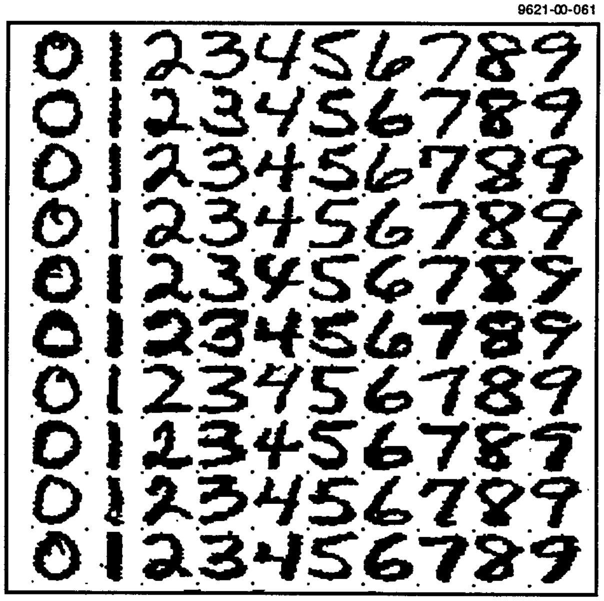


Figure 22. Examples of handwritten digit exemplars from the Post Office database used for optical neural network learning.

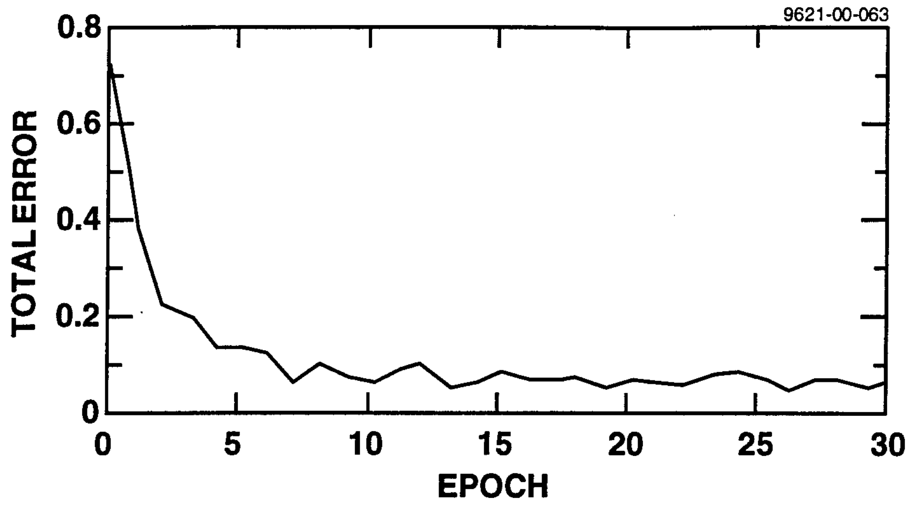


Figure 23. Optical Perceptron learning curve for the handwritten digit problem. *The network consisted of 676 input neurons and 10 output neurons. Eight examples of each of the ten digits for a total of 80 exemplars were used in the learning phase. The network achieved a 75% correct classification rate on 40 test digits not used in the learning phase.*

where E is the total error, $D_{i,c}$ is the desired value for output neuron i given exemplar input c, and $y^{(n)}{}_{i,c}$ is the actual output for exemplar c. The weight adjustment rule during learning is based on the steepest descent rule:

$$\Delta w_{i,j}^{(n)} = -\eta \frac{\partial E}{\partial w_{i,j}^{(n)}} \tag{31}$$

where the superscript (n) refers to the weight layer being updated. Assuming a two-layer network, and without going into the details of the derivation here, the error gradient is given by the following set of equations for the output layer (see the references for details):

$$\begin{aligned} &\frac{\partial E}{\partial w_{i,j}^{(2)}} = \delta_i^{(2)}\, \mathrm{y}_\mathrm{j}^{(1)}, \\ &\delta_i^{(2)} = \left(y_i^{(2)} - D_i\right) g\left(\mathrm{y}_\mathrm{i}^{(2)}\right), \\ &g(y) = f'\left[f^{-1}(y)\right] = (1+y)\,(1-\mathrm{y}) \end{aligned} \tag{32}$$

and by these equations for the hidden layer:

$$\begin{aligned} &\frac{\partial E}{\partial w_{j,k}^{(1)}} = \delta_j^{(1)} y_k^{(0)}, \\ &\delta_j^{(1)} = g(y_j^{(1)}) e_j^{(1)}, \\ &e_j^{(1)} = \sum_i \delta_i^{(2)} w_{ij}^{(2)} \end{aligned} \tag{33}$$

where $f(x)=(1-e^{-x})/(1+e^{-x})$ is the neuron sigmoidal response function for neuron values between -1 and 1.

One possible issue in the optical implementation of backpropagation is its sensitivity to the accuracy of representation of the functions f() and g(). Due to inherent nonlinearities and nonuniformity in present SLMs, it may not be possible to represent f() and g() in exactly the above form. We have performed a sries of computer simulations of backpropagation with slightly different f()s and g()s. We found that the network performance was relatively insensitive to the precise values of these functions as long as the general sigmoidal and hump-like forms of f() and g(), respectively, were preserved.

In the optical system we actually implemented a variation of backpropagation in which the input error signals were trinary quantized to +1, 0, and -1 according to the algorithm reported by Shoemaker et al.[50] They found that trinary quantization improved the convergence speed of backpropagation for a wide variety of problems. Our own computer simulations confirmed this. However, our primary reason for trinary quantization was to avoid amplitude-dependent phase errors in our liquid-crystal based SLM.

The optical input plane for backpropagation is shown in Figure 24. The network consisted of three neuron layers (L_0, L_1, and L_2) and two weight layers. Note that the L_1-L_2 weights are actually implemented as two separate sets of weights, one for the forward pass L_1-L_2 and another one for the backward pass L_2-L_1. As explained previously, this is done because the fanned reference beam (R plane) is always used to read out the unfanned object beam (O plane). In order to implement forward and backward passes through the same set of effective weights, two sets of photorefractive weights must be exposed. In this case they are exposed so as to make the forward and backward weights as nearly equal as possible (symmetric connections, $w_{ij}=w_{ji}$) although they can also be made different (asymmetric connections) if required for certain networks. Although the L_1-L_2 and L_2-L_1 sections of the input plane are shown spatially separated for clarity, in actuality they are spatially interleaved in order to make the connections as symmetric as possible. The solid arrows indicate optical connections between neurons via the hologram. Dashed arrows denote electronic transfer of detected outputs from one layer (O plane) to the inputs of the next layer in the R plane. This electronic operation is an order N operation while the optical interconnection is order N^2, where N is the number of neurons. The forward and backward weights between layers L_1 and L_2 were adjusted simultaneously in the interests of keeping the forward and backward weights as equal as

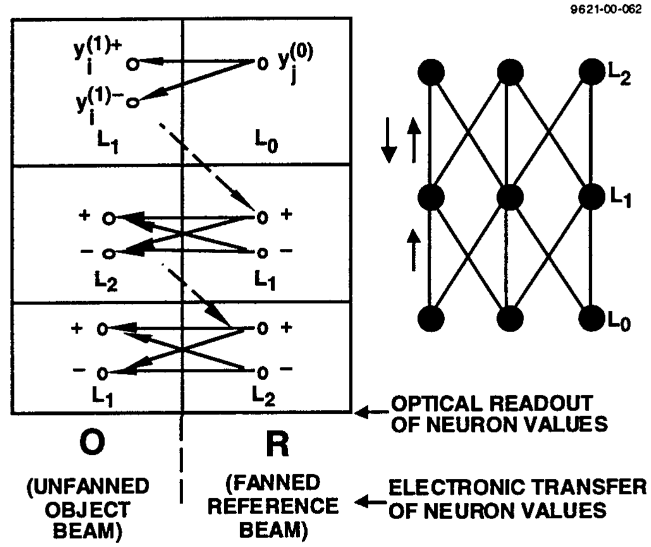


Figure 24. Optical input plane for backpropagation with a single hidden layer.

possible. This also causes unnecessary self-connections to be formed. The self-connections do not, however, affect the operation of the network because they do not contribute to the inputs of one layer to the next. The simultaneous adjustment of forward and backward weights also speeds up execution.

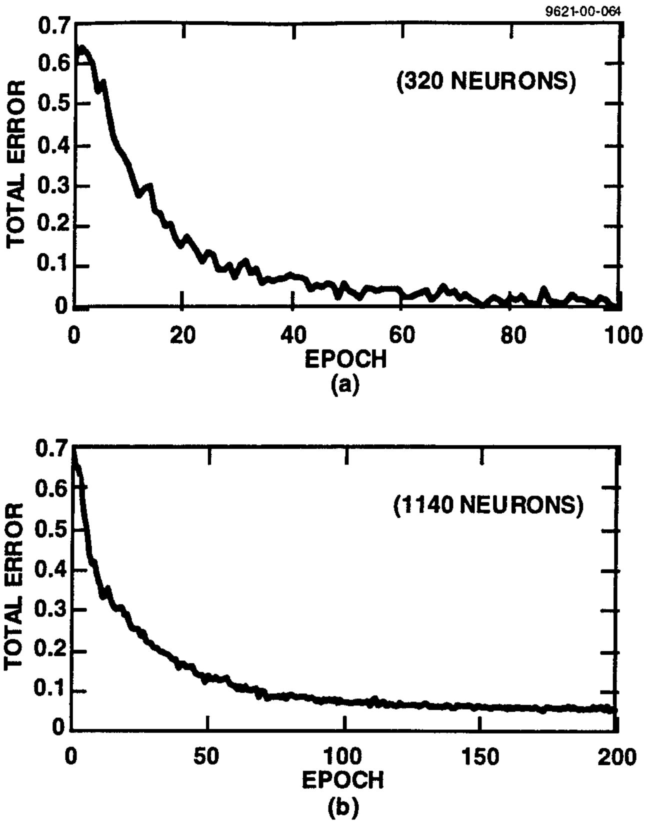


Figure 25. Optical backpropagation learning of two random input/output associations. *Results for two networks are shown, one with 320 neurons and the other with 1140 neurons.*

Experimental results for the problem of transforming one random binary pattern into another using optical backpropagation are shown in Figure 25. We plot two curves of total output error versus epoch during learning for two different-sized networks. In both cases the network consisted of three layers of neurons with a single hidden layer and the network learned to associate two pattern pairs. In Figure 25(a) and (b) the numbers of neurons in the three layers were 160-80-80 and 570-285-285, respectively. In Figure 26 we show the behavior of the optical neurons during learning and readout. In this case 252 neurons were arranged in three layers as 128-62-62. As shown in Figure 26(a), for two exemplars the output error decreased to 0 after 75 epochs. In order to track the evolution of the output pattern for individual input patterns during learning, we plot the inner-product of the output pattern vector with the reference vector (1,1,1,...,1) in Figure 26(b) for each of two input patterns. This projection reduces the dimensionality of the output to one so that it can be plotted. We can clearly see the buildup of the weights from small initial values to different steady state values for the two exemplars. In Figure 26(c) we plot the output pattern vector projections versus epoch as the network is continuously read out after learning is completed. The weight decay can be clearly seen. Nevertheless, the error rate is less than 3% after more than 500 readout epochs, despite weight decay, as is shown in Figure 26(d). This corresponds to 1000 separate readouts of the network. The output error does not increase until after the weights have decreased by a large amount, indicating the nonlinearities present in the network allowed it to learn a “safety margin” which gave it limited immunity to weight decay.

We applied the optical backpropagation network to the same handwritten digit recognition problem discussed in the Perceptron section above. In order to improve performance we encoded each of the class labels for the ten digits using the the first ten rows of a 16 column Hadamard matrix. The advantages of a Hadamard representation include orthogonality and roughly balanced numbers of 1’s and -1’s in the exemplar outputs. This balancing results in a more uniform distribution of weight

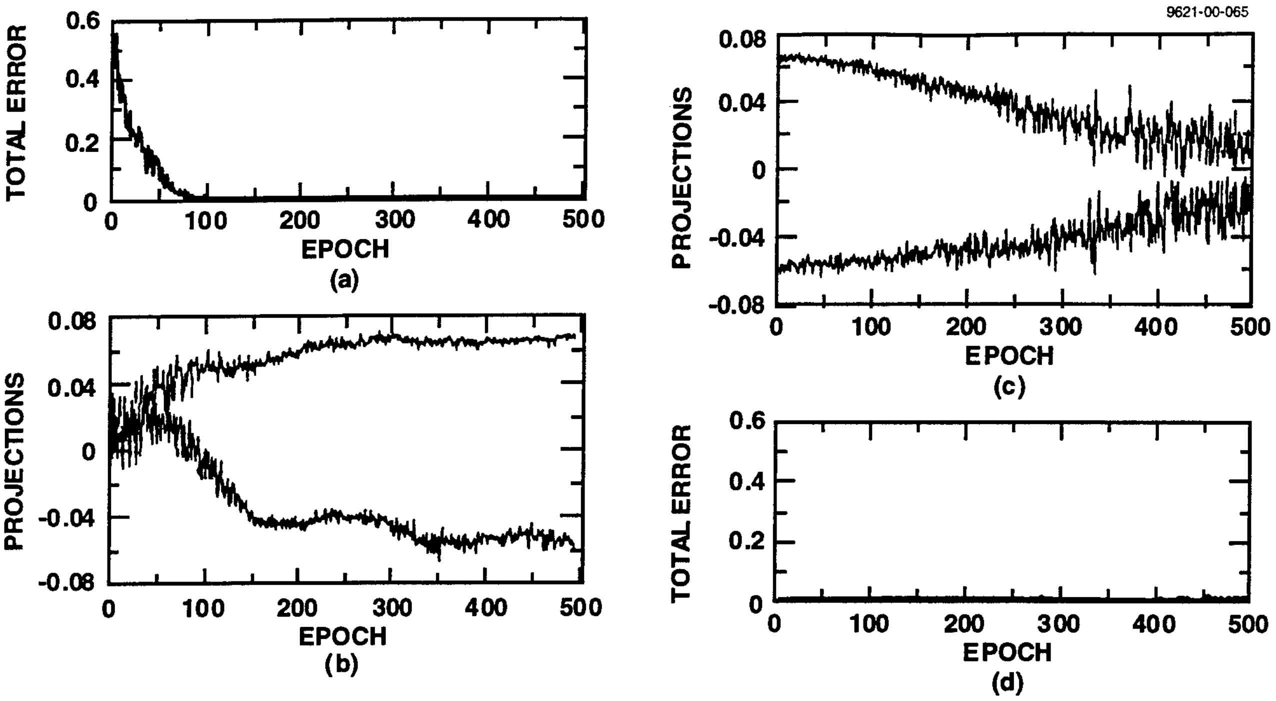


Figure 26. Behavior of optical neurons during backpropagation learning and readout. *(a) Total error versus epoch during learning, (b) projection of output layer values onto a reference vector during learning. Separate curves for two exemplar input patterns are shown, (c) output projections for the two exemplar inputs during continuous readout of the network after learning was completed. The network was read out 500 times with each of the two exemplars for a total of 1000 readouts. Note the decrease in projection magnitudes due to weight decay, (d) corresponding errors during the readout phase. The neuron nonlinearities kept the error rate small for more than 1000 readouts despite weight decay.*

values which reduces the optical dynamic range requirements. In reading out the network, vector quantization on the output layer was performed in which the Hadamard vector closest to the network output was selected as the class label. The network consisted of 676 input neurons (the handwritten digit patterns consisted of 26 × 26 pixels), 90 hidden neurons, and 16 output neurons.

The error curve during learning for the case of two classes with 20 exemplars each is shown in Figure 27(a). A correct classification rate of 97% was achieved on test digits not used for learning, showing that generalization occured. The performance decreased when the number of classes was increased to ten, as can be seen in Figure 27(b). During learning, the errors first reached zero after 200 epochs, but an erratic component kept the average error rate at 10%. After terminating learning at 800 epochs, the network achieved a correct recognition rate of 55% on 20 test digits not used during learning. This is still much greater than the 10% expected by chance, indicating that some generalization occured. Computer simulations using the same number of exemplars achieved a 63% correct recognition rate. Computer simulations using 2000 exemplars achieved a 97% correct recognition rate, confirming that large numbers of exemplars are desirable. (Note, however, that 2000 exemplars is still much smaller than what would be predicted as necessary using the worst-case estimates of Section 2.2 for a network of this size.) Our current research is focused on improving these initial results by increasing the number of exemplars and classes that can be learned and improving the recognition rate. By so doing, nonfundamental factors limiting the performance can be identified and eliminated.

5.3 Self-Organizing Map

A schematic illustration of the operation of a Kohonen self-organizing map (SOM) neural network[51] is shown in Figure 28. The function of the SOM is to map "close" points in the input space to "close" points in a lower dimensionality output space. The applications for SOM include data compression and pattern recognition. Results for a SOM neural network implemented using SPONN are shown in Figure 29. In this experiment the input layer contained 676 neurons arranged in a 2-D format. The output layer contained 512 neurons, resulting in a total of 3.5×10^5 weights. The training set consisted of handwritten digits from a Postal Service database. In Figure 29 we show the input plane and the corresponding output plane before and after unsupervised learning. The standard SOM algorithm was implemented with linear time shrinkage of the update neighborhood centered on the maximum responding output neuron. Nearest neighbors were defined on a 2-D mesh in the output plane. After learning was completed, the optical SOM clustered the inputs into classes corresponding to numerals, as shown in Figure 29.

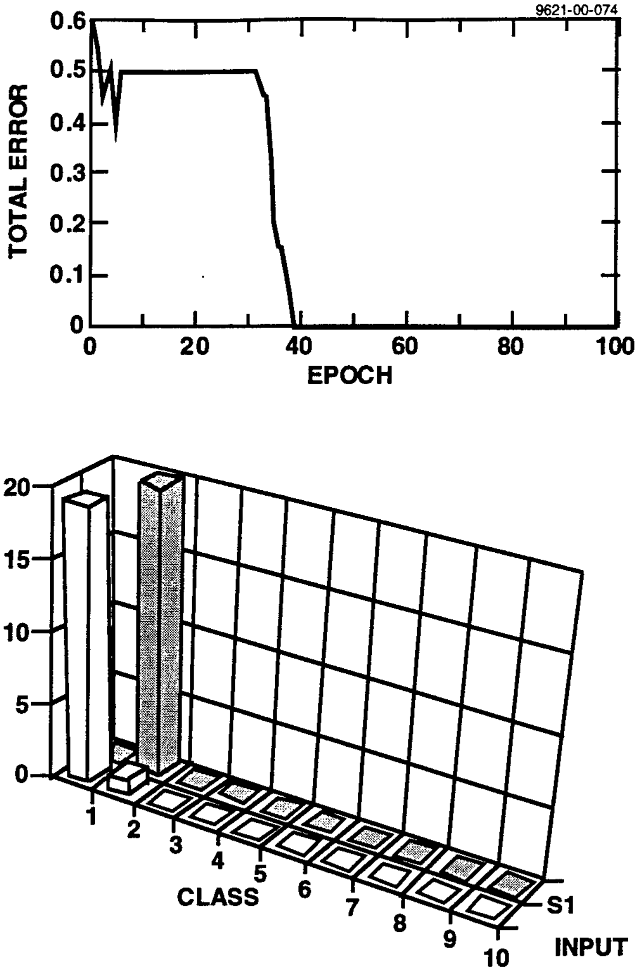


Figure 27. Optical backpropagation learning of handwritten digits. *Network consisted of 676 input neurons, 90 hidden neurons, and 16 output neurons. (a) Two classes and 10 exemplars per class. Generalization performance on novel test digits was 97% correct classification, (b) Ten classes and 1 exemplar per class. Generalization performance on novel test digits was 55% correct classification.*

5.4 Future Directions

Our experience in constructing and operating an optical neurocomputer has revealed to us two general types of issues related to implementing large neural networks. They include issues related to the training of large networks (independent of the particular implementation architecture) and issues specifically related to improving the performance of HONNs.

With regard to training, generally speaking, large fully connected neural networks will require very large training sets and long training times (Section 2.2) unless mitigating measures are taken. These measures include limiting the connectivity, incorporating hints and prior knowledge in the structure of the network, using self-organization to reduce the dimensionality of input patterns, and

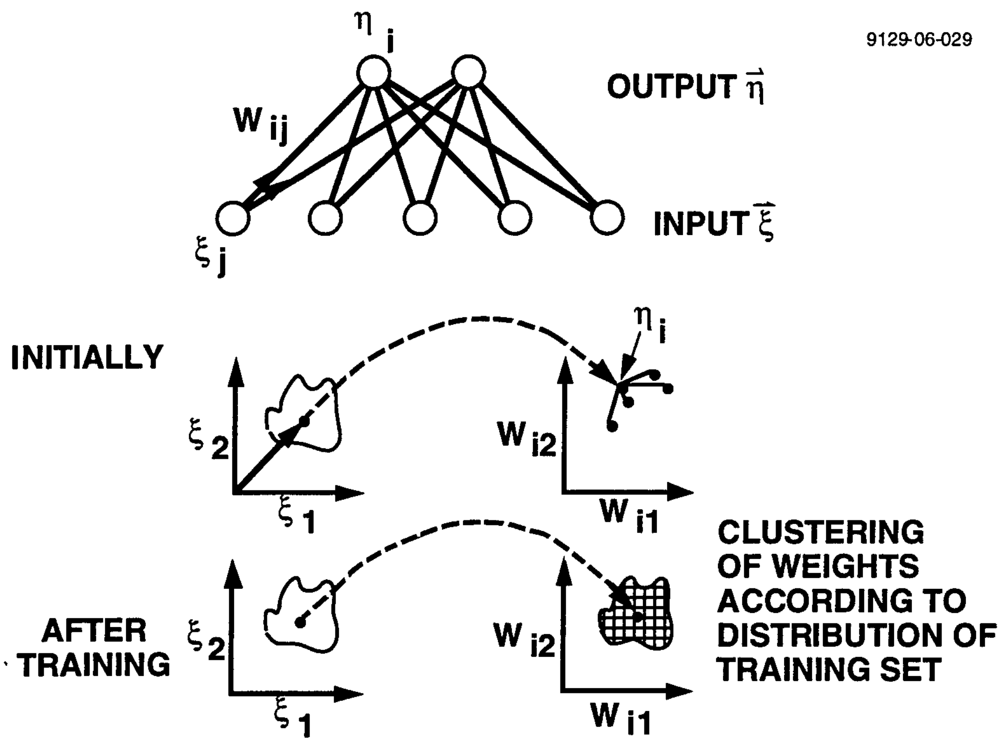


Figure 28. Results for optically-implemented Kohonen self-organizing neural network. *(a) Before training (b) After training.*

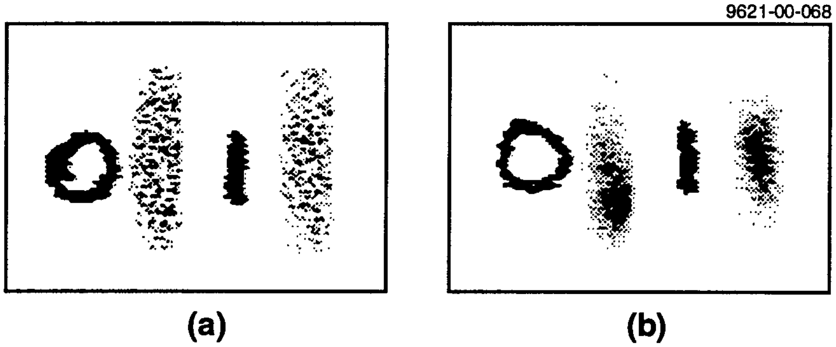


Figure 29. Optical implementation of partially connected neural networks.

using regularization to eliminate unimportant weights. An important capability which has not yet been implemented in optical neurocomputers is programmability of the neural network connectivity. In the SPONN neurocomputer built under this contract, all neurons in one layer can potentially form connections with all neurons in the next layer. We know that biological neural networks do not exhibit such global connectivity. Typically, neurons are connected to other nearby neurons. For example, neurons in the visual cortex respond to features within receptive fields which are a small fraction of the total visual input. Such compartmentalization serves to organize the nervous system and reduce its complexity, which makes learning more tractable. Programmable connectivity is important to reducing the size of training sets required for large neural networks.

One method we propose for controlling network connectivity is to control the mutual coherence of neurons by time-modulating their outputs using orthogonal or near-orthogonal phase codes, as shown in Figure 30. Hebbian weight updates between neurons with dissimilar phase codes will average out to zero, while updates between neurons with identical phase codes will build up to a nonzero value. Such phase modulation of neurons can be implemented optically using the setup shown in Figure 31. Neuron activity values are displayed on SLM 1. SLM 2, operating in transmission mode, displays regions which are time-modulated by different phase codes. Light beams connecting neurons are then amplitude-modulated by activity values on SLM 1 and phase-modulated in time by phase codes on SLM 2 which define what combinations of neuron groups will form connections. The phase of light fringes formed by any two optical beams will vary in time if the phase codes of the beams are different, therefore any photorefractive gratings in the hologram will be smeared out and no connection will be formed. If the phase codes are equal, the fringes will be stationary, allowing photorefractive gratings and the resultant connection to be formed. In this way, large neural networks with limited connectivity can be formed.

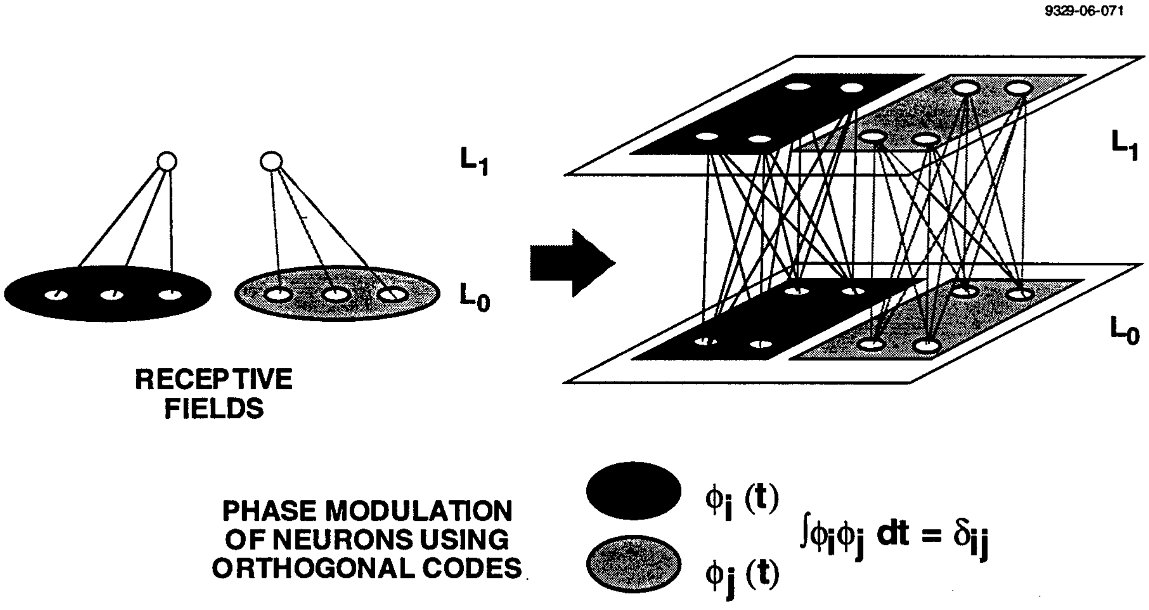
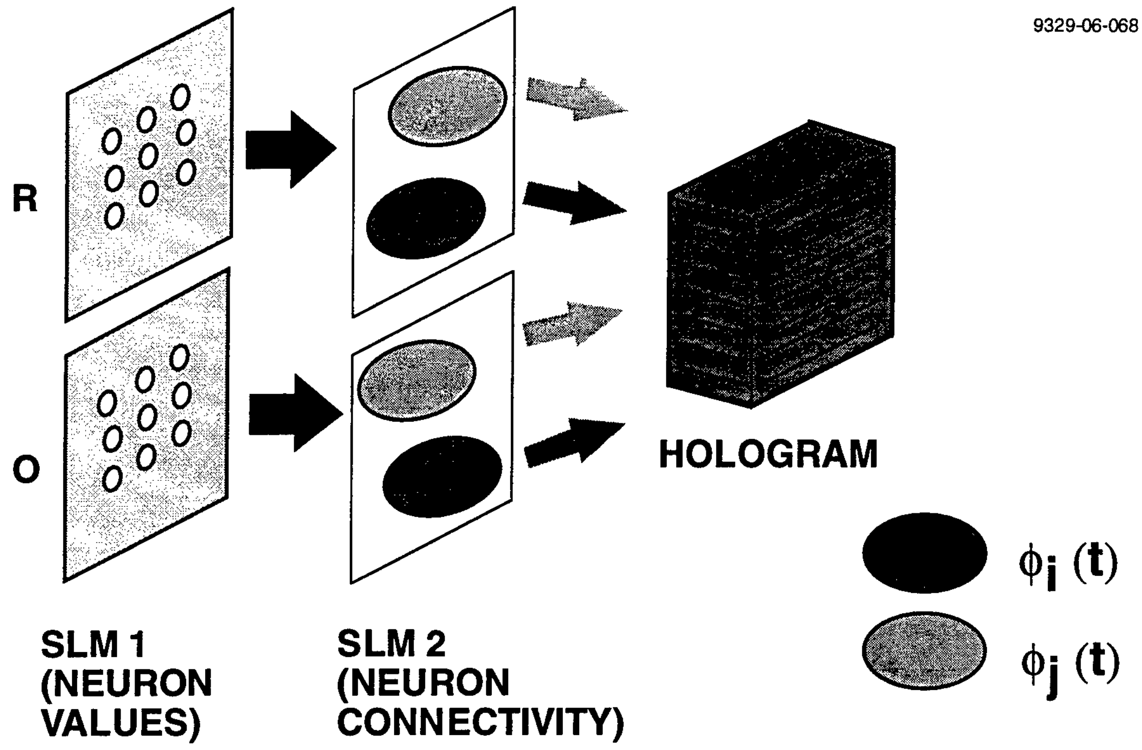

9329-06-071
L_1
L_0
RECEPTIVE FIELDS
L_1
L_0
PHASE MODULATION OF NEURONS USING ORTHOGONAL CODES
$\phi_i(t)$
$\phi_j(t)$
$\int \phi_i \phi_j \, dt = \delta_{ij}$

Figure 30. Partial connectivity can be programmed in optical neural networks by controlling the mutual coherence of optical neurons.

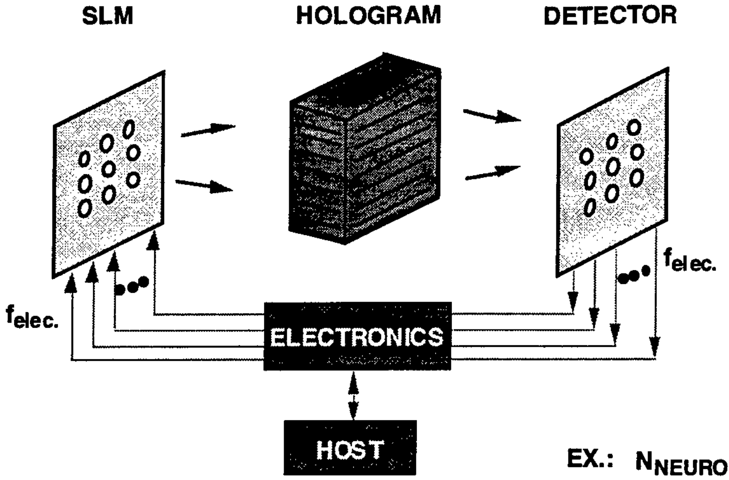


$\phi_i(t)$ AND $\phi_j(t)$ ARE ORTHOGONAL PHASE CODES: $\int \phi_i \phi_j \, dt = \delta_{ij}$

Figure 31. Optical neurocomputer based on smart pixels.

An important factor in determining the overall system performance of the optical neurocomputer is how information is transported from the optical detectors (CCD pixels) back to the optical neurons (SLM pixels). In the present HONN system, this communication is done serially over the bus of the host computer, as shown in Figure 32. This pathway can be a potential bottleneck for the system. As shown in the figure, the computation rate for a bus-limited HONN is proportional to the number of neurons for fully connected networks. It is also proportional to the input/output bandwidth of the detector and SLM. If we assume 10^5 fully connected neurons and a detector bandwidth of 10^7 neuron values per second, then a throughput of 10^{12} connections per second could be theoretically achieved. A potential development path is to incorporate a fully parallel optical interconnection between the detector array and the SLM. This could be done by incorporating emitters, detectors, and perhaps some simple processing electronics at each pixel location. Such "smart pixels"[52] could greatly increase the throughput of the HONN, as shown in Figure 33. By eliminating the bottleneck caused by using the host bus to transfer information from detectors to the optical neurons, the throughput becomes proportional to the number of weights rather than the number of neurons. The theoretical throughput rate increases to 10^{17} connection updates per second.

9329-06-070

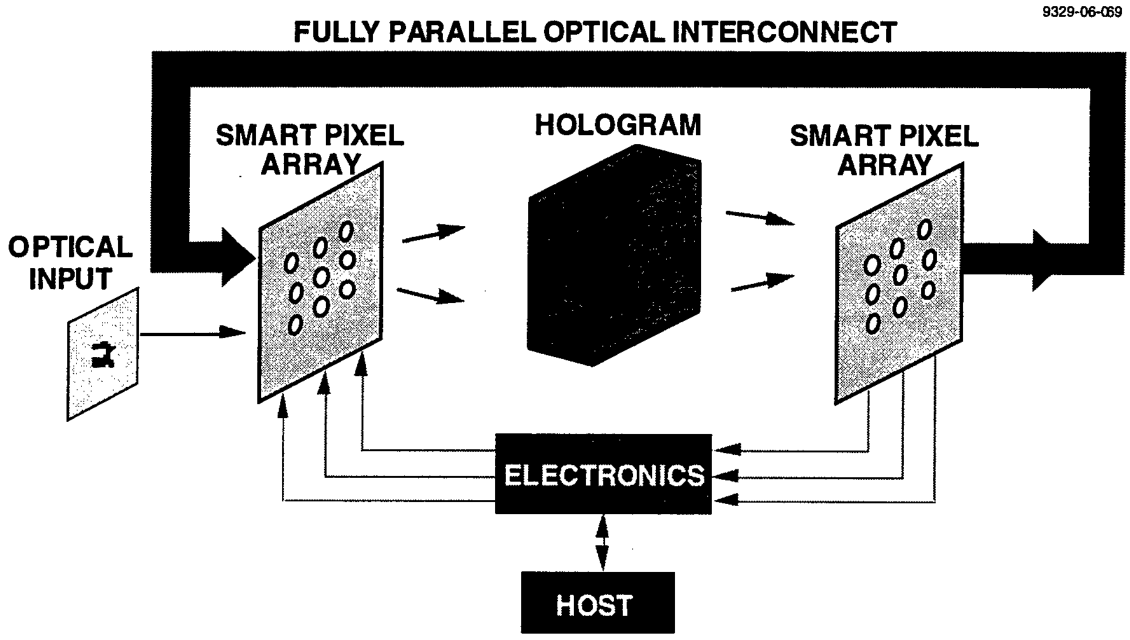


COMPUTATION RATE: $R = \frac{N_{WEIGHTS}}{T_{FRAME}} = \frac{N_{WEIGHTS}\ N_{I/O}\ f_{elec.}}{N_{NEURONS}}$

EX.: $N_{NEURONS} = 10^{10}$
$N_{WEIGHTS} = 10^{10}$
$N_{I/O}\ f_{elec.} = 10$ MHZ
→ $R = 10^{12}$ CONN./SEC.

⇒ FOR FULLY CONNECTED NETWORKS,SPEEDUP IS LINEAR IN $N_{NEURONS}$

Figure 32. Parallelism of smart pixel based holographic neural network.



COMPUTATION RATE: $R = \frac{N_{WEIGHTS}}{T_{FRAME}}$, EX.: $N_{WEIGHTS} = 10^{10}$
$T_{FRAME} = 100$ nsec
→ $R = 10^{17}$ CONN./SEC.

SPEEDUP IS LINEAR IN $N_{WEIGHTS}$

Figure 33. Development paths of holographic neurocomputer.

Two potential development paths for SPONN are shown in Figure 34. If the currently used video-based image processing cards are kept for input/output to the optical portion of SPONN, then future improvements will come from increasing the neuron density on the SLM and CCD detector array. The frame times will stay roughly constant and improvements will follow the lower slope shown. This development path assumes commercially available video display devices based on liquid crystals, which have response times of about 30 msec, are used as the SLM. Incorporating smart pixels will result in greater increases in performance along the steeper slope, but at greater risk since we could not then leverage the large amount of research being done on multi-media display devices.

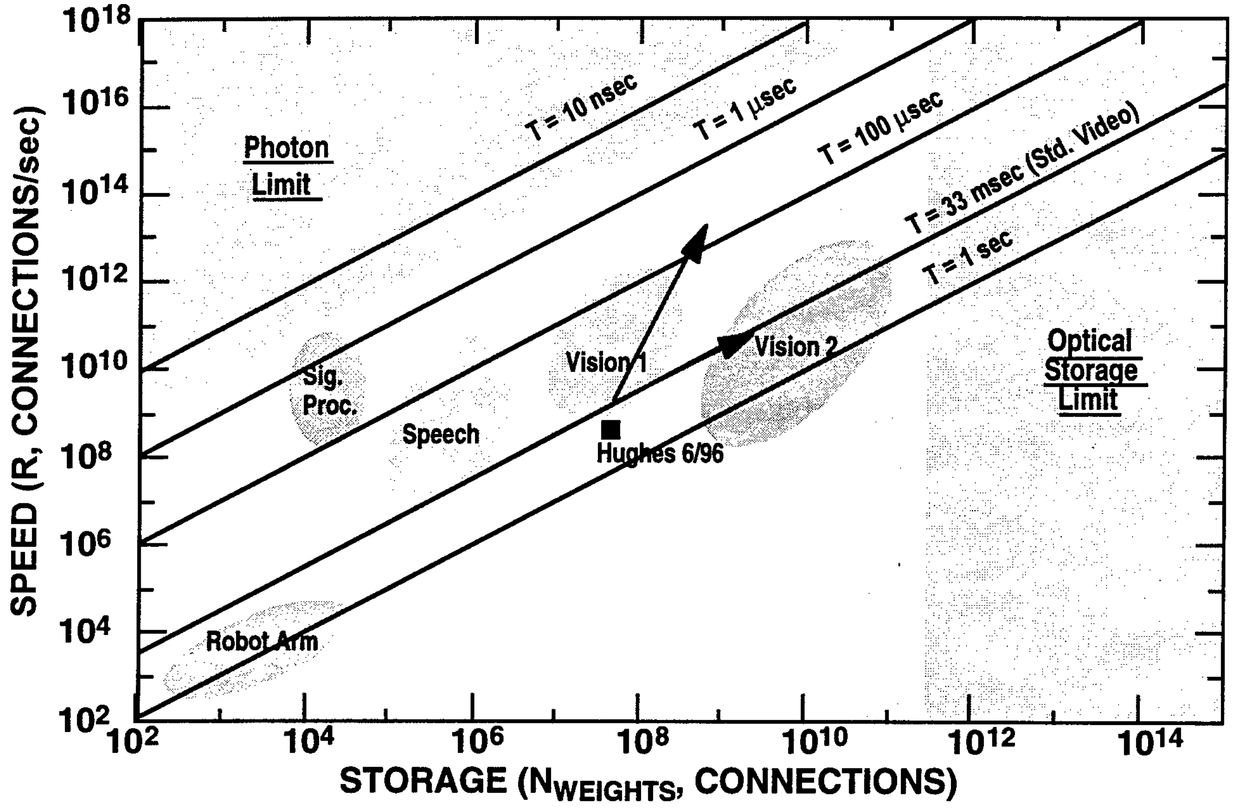


Figure 34. Holographic optical neural network development paths.

6. SUMMARY

In this final report we described a compact optical neurocomputer based on a laser diode light source and holographic storage and optical interconnection using photorefractive crystals. This neurocomputer utilizes a novel technique for optically implementing neural network models: multiple grating holography. By distributing each connection weight among a plethora of cascaded gratings which vary in position and orientation, several advantages are obtained. First, Bragg degeneracy is eliminated which permits the placement of pixels in arbitrary 2-D patterns in the input and output planes, leading to new flexibility in holographic recording. Second, distortions due to beam coupling are greatly reduced which results in undistorted weight vectors and good quality reconstructed images. This permits the design of compact programmable holographic neurocomputers which are composed of a single photorefractive crystal, SLM, and detector. The utilization of the SLM is maximized because every pixel can be used without fear of crosstalk. In this report we described the construction of a compact neurocomputer based on a laser diode light source and this innovative holographic recording method. We showed the programmability of such an optical neurocomputer by demonstrating several networks (including multi-layer backpropagation) on the same hardware without adjustment of the optical components and with throughput rates of up to 10^8 weight updates per second during learning. A new coherent detection method for efficient optical representation of bipolar weights and neurons was also developed. A technique for programming the connectivity of neural network models implemented optically was invented as well under this contract.

As discussed in Section 6, we see future optical neurocomputers being used for applications which require large neural networks. Analog optics has advantages over electronics in terms of power consumption, size, and cost as the neural network size increases. The challenge for electronics is not the number of neurons, but rather how to fit the interconnections between neurons into the 2-D format of silicon chips and wafers. The benefits that optics brings to neural networks include massive parallelism and true three-dimensional interconnectivity. A real-time hologram can interconnect two 2-D planes of neurons in arbitrary patterns using the third dimension. This makes optics especially suitable for implementing large highly-interconnected neural networks (10^3 to 10^6 neurons) with high processing rates (10^{10} to 10^{12} connections/sec). The advantages of optics over electronics become less as the network size decreases. Specialized electronic chips and parallel computers will be preferred for problems with smaller numbers of neurons.

We saw in Section 3.1 that a feature of real-time holography using photorefractive crystals is weight decay, the partial erasure of previously recorded weights that is incurred as the weights are updated or read out. Weight decay has both positive and negative aspects. On the positive side, weight decay can help to regularize a network. Unused or weak weights decay away so the final network is simpler in form than it would be otherwise, which improves generalization. It also prevents weights from becoming too large and saturating the outputs. On the other hand, weight decay can distort learning rules, although the distortion can be minimized by using small weight values and increasing the detector gain. Another effect of weight decay is that the final weight value is equal to the *average* of all the weight updates, rather than the simple sum. This limits the number of exposures that can be recorded in the crystal to a finite value which is dependent on the dynamic range of photorefractive gratings. As discussed in Section 3.1, this number can be quite large, as the photorefractive dynamic range has been measured to be at least 100 db. Theoretical calculations indicate the limiting dynamic range may be 140 db. This suggests that up to 10^7 separate exposures can be superimposed in photorefractive crystals. Nevertheless, the number of exemplars required for good generalization may also be large, especially for large networks. Methods may have to be developed to reduce the number of required exemplars, such as providing "hints" or tailoring the network structure to the problem at hand.

The laser diode-based optical neurocomputer we have described in this final report provides further experimental evidence that it is technically feasible to implement large neural networks using real-time volume holography in photorefractive crystals. Future generation optical neurocomputers will be economically successful in applications requiring very large neural networks, such as vision, database mining, and optimization, but only if several issues are addressed and resolved by continued research. It is well known that large neural networks, although powerful, can require large training sets and long training times, especially if existing small neural network training algorithms are simply scaled up to train large networks. Techniques for reducing the training set size and training times, such as regularization (which can utilize weight decay), pruning, prestructured connectivity, modularization, and hints, will need to be incorporated in optical neural networks to make them practical. In addition, we feel that self-organization will play an important role in future optical neural networks as a means to handle the increased complexity of large neural networks.

In summary, holographic optics based on photorefractive crystals appears to be extremely promising for the implementation of large neural network models. Working optical neural networks have been demonstrated even though the field is still on the early part of the learning curve. Based on these early results, we are optimistic that optics will play a significant role in future implementations of neural networks.

7. REFERENCES

1. M. L. Minsky and S. A. Papert, *Perceptrons*, M.I.T. Press, 1969.
2. V. Y. Kreinovich, "Arbitrary Nonlinearity is Sufficient to Represent All Functions by Neural Networks: A Theorem," Neural Networks Vol. 4, pp. 381-383 (1991).
3. The performance figures of the electronic hardware were taken from J. Alspector, "Parallel Implementations of Neural Networks: Electronics, Optics, Biology," in *Technical Digest of 1991 Topical Meeting on Optical Computing*, Optical Society of America, 1991.
4. H.-Y. Li, Y. Qiao, and D. Psaltis, "An optical network for real time face recognition," Appl. Opt., March, 1993.
5. F. Mok and H. M. Stoll, "Holographic inner-product processor for pattern recognition," SPIE Proceedings Vol. 1701, 1992.
6. S. Judd, "Learning in Networks is Hard," Proceedings of IEEE International Conference on Neural Networks, San Diego, 1987, p. II-685.
7. *Neural Networks and Natural Intelligence*, edited by S. Grossberg, M.I.T. Press (1988).
8. T. Kohonen, *Self-Organization and Associative Memory* (Second Edition), Springer-Verlag (1988).
9. D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter, "The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function," IEEE Trans. Neural Networks **1**, 296-298 (1990).
10. V. N. Vapnik and A. Y. Chervonenkis, "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities," Theory of Probability and Its Applications **16**, 264-280 (1971).
11. J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation,* Addison-Wesley, Redwood City, 1991.
12. E. B. Baum and D. H. Haussler, "What Size Net Gives Valid Generalization?," Neural Computation **1**, 151-160.
13. J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation,* Addison-Wesley (1991).
14. Y. Abu-Mostafa, "Hints," Neural Computation 7, 639-671 (1995).
15. Y. Abu-Mostafa, "Appendix D: Complexity in Neural Systems," in *Analog VLSI and Neural Systems* by C. Mead, Addison-Wesley, 1989.
16. C. L. Giles and T. Maxwell, Learning, invariance, and generalization in high-order neural networks, Appl. Optics **26**, 4972-4978 (1987).
17. M. Cohen, H. Franco, N. Morgan, D. Rumelhart, and V. Abrash. Context-dependent multiple distribtion phonetic modeling, in *Advances in Neural Information Processing Systems 5,* D. Touretzky, Ed., Morgan Kaufmann, 1993.
18. B. Widrow, D. E. Rumelhart, and M. A. Lehr. Neural networks: applications in industry, business, and science, Comm. of the Assoc. for Computing Machinery **37** (3), 93-105 (1994).

19. D. Psaltis, D. Brady, and K. Wagner, “Adaptive optical networks using photorefractive crystals,” Appl. Opt. **27**, 1752-1759 (1988).

20. B. K. Jenkins and A. R. Tanguay, Jr., “Photonic Implementations of Neural Networks,” in *Neural Networks for Signal Processing*, B. Kosko, ed., Prentice-Hall, 1992, 287-382.

21. D. Psaltis, D. Brady, X.-G. Gu, and S. Lin, “Holography in Artificial Neural Networks,” Nature **343**, 325-330 (1990).

22. See special issue of Applied Optics on neural networks, March, 1993.

23. H. Kogelnik (1969). “Coupled wave theory for thick holograms,” Bell Sys. Tech. J. **48**, 2909-2946.

24. F. H. Mok and H. M. Stoll (1992). “Holographic Inner-Product Processor for Pattern Recognition,” SPIE Proceedings **1701**.

25. N. V. Kukhtarev, V. Markov, and S. Odulov, “Transient energy transfer during hologram formation in $LiNbO_3$ in external electric field,” Opt. Comm. **23**, 338-343 (1977).

26. Y. Taketomi, J. E. Ford, H. Sasaki, J. Ma, Y. Fainman, and S. H. Lee, “Incremental recording for photorefractive hologram multiplexing,” Opt. Lett. **16**, 1774-1776 (1991).

27. K. Blotekjaer, “Limitations on holographic storage capacity of photochromic and photorefractive media,” Appl. Opt.

28. T. Y. Chang, J. H. Hong, F. Vachss, and R. McGraw, “Studies of the Dynamic Range of Photorefractive Gratings in Ferroelectric Crystals,” J. Opt. Soc. Am. B **9**, 1744-1751 (1992).

29. C. Gu and P. Yeh, “Scattering due to randomly distributed charge particles in photorefractive crystals,” Opt. Lett. **16**, 1572-1574 (1991).

30. D. Psaltis, D. Brady, X.-G. Gu, and S. Lin, “Holography in artificial neural networks,” Nature **343**, 325-330 (1990).

31. K. Rastani and W. M. Hubbard, “Large interconnects in photorefractives: grating erasure problem and a proposed solution,” Appl. Opt. **31**, 598-605 (1992).

32. C. Slinger, “Analysis of the N-to-N volume holographic neural interconnect,” J. Opt. Soc. Am. A **8**, 1074-1081 (1991).

33. G. P. Nordin, *Volume Diffraction Phenomena for Photonic Neural Network Implementations and Stratified Volume Holographic Optical Elements,* Ph.D. Thesis, University of Southern Calif. (1992).

34. A. Chiou, “Anisotropic cross talk in an optical interconnection by using a self-pumped phase-conjugate mirror at the Fourier plane,” Opt. Lett **17**, 1018-1020 (1992).

35. Y. Owechko, “Self-Pumped Optical Neural Networks,” in *Proceedings of 1989 Topical Meeting on Optical Computing*, (Optical Society of America, 1989 Technical Digest Series Vol. 9), 44-47.

36. Y. Owechko, in *Conference Record of 1990 International Topical Meeting on Optical Computing*, (SPIE, 1990) pp. 142-146.

37. Y. Owechko and B. H. Soffer (1991). "Optical interconnection method for neural networks using self-pumped phase conjugate mirrors," Opt. Lett. **16**, 675-677.

38. G. J. Dunning, Y. Owechko, and B. H. Soffer (1991). "Hybrid optoelectronic neural networks using a mutually pumped phase conjugate mirror," Opt. Lett. **16**, 928-930.

39. Y. Owechko and B. H. Soffer, "Optical Neural Networks Based on Liquid Crystal Light Valves and Photorefractive Crystals," SPIE Proceedings, Vol. 1455, 136-144 (1991).

40. M. D. Ewbank, "Mechanism for photorefractive phase conjugation using incoherent beams," Optics Letters **13**, 47-49 (1988).

41. *Photorefractive Materials and Their Applications I*, edited by P. Gunter and J. P. Huignard, Springer-Verlag (1988).

42. J. E. Ford, Y. Fainman, and S. H. Lee, "Enhanced photorefractive performance from 45°-cut $BaTiO_3$," Appl. Opt. **28**, 4808-4815 (1989).

43. See special issue of IEEE Journal of Quantum Electronics on Semiconductor Diode-Pumped Solid-State Lasers, Vol. 28, No. 4 (1992).

44. B. A. Wechsler, M. B. Klein, C. C. Nelson, and R. N. Schwartz, "Spectroscopic and photorefractive properties of infrared-sensitive rhodium-doped barium titanate," Opt. Lett. **19**, 536-538 (1994).

45. J. H. Hong, S. Campbell, and P. Yeh, "Optical pattern classifier with Perceptron learning," Appl. Opt. **29**, 3019-3025 (1990).

46. R. Rosenblatt, *Principles of Neurodynamics,* Spartan Books, New York, 1962.

47. D. B. Parker, "Learning Logic," Invention Report S81-64, File 1, Office of Technology Licensing, Stanford Univ. (Oct. 1982).

48. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing, Vol. 1*, D. E. Rumelhart and J. L. McClelland, Eds., MIT Press, (1986).

49. P. A. Shoemaker, M. J. Carlin, and R. L. Shimabukuro, "Backpropagation Learning With Trinary Quantization of Weight Updates," Neural Networks **4**, 231-241 (1991).

50. T. Kohonen, *Self-Organization and Associative Memory* (Second Edition), Springer-Verlag (1988).

51. See special issue of IEEE Journal of Quantum Electronics on smart pixels, February, 1993.

Appendix A
DESCRIPTION OF HONN SOFTWARE

A.1 List of SPONN Programs

	Register Address	Frame Memory Address
VFG 1 (Input)	0x360	0xD0000
VFG 2 (Output and VisionBus Master)	0x320	0xD0000
IPA	0x340	0xD0000

Program	Purpose	Comments
learn22.c	Single output Perceptron, dual-rail detection	
learn22e.c	Single output Perceptron, balanced coherent detection	Compatible with Cohu variable integration CCD camera
learn29.c	Hologram superposition test	
learn31.c	Multiple-output Perceptron	
learn34q.c	Backpropagation classification of handwritten digits, dual-rail detection	
learn34s.c	Backpropagation classification of handwritten digits, balanced coherent detection	Compatible with Cohu variable integration CCD camera
learn40.c	Self-organizing classification of handwritten digits	Compatible with Cohu variable integration CCD camera
learn41e.c	Tests weight encoding using dual-rail method, records readout of Hadamard vectors	
learn41f.c	Tests weight encoding using dual-rail method, records time evolution of weights	
coher1.c	Tests weight encoding using coherent detection, records time evolution of weights	
coher2.c	Tests weight encoding using balanced coherent detection, records time evolution of weights	Compatible with Cohu variable integration CCD camera
weight.c	Tests hologram fan-in	
init1.c	Initializes VFG 1	
init2.c	Initializes VFG 2	
init3.c	Initializes IPA	
alignvfg.c	Aligns output pixels	Generates coordinate and normalization file for output neurons (align64.dat). Works with Cohu variable integration CCD camera.
rneuron.c	Plots selected output neuron value vs time	Works with Cohu variable integration CCD camera.
noise.c	Plots selected pixel values vs time	Works with Cohu variable integration CCD camera.

A.2 Optical Neurocomputer Program Listing: learn34s.c (Backpropagation)

```
/* learn34s.c
  Optical backpropagation using balanced coherent detection to represent bipolar
  outputs. All neurons are bipolar except for input neurons.

  Values of selected neurons are recorded during learning in file
  lrn34s.csv. Output coobdinates and gain and offset correction factors are
  read from align64.dat, which is generated using alignvfg.c.
  No. of random patterns used during weight initialization is an input
  parameter.

  This program implements a two-layer, single hidden-layer,
  backpropagation neural network implemented in a cascaded-grating optical
  neural network which recognizes handwritten digits
  from a Post Office database. The network has 676 unipolar (0,1) input
  neurons, 540 bipolar (-1,1) hidden neurons, and 32 bipolar (1,-1)
  output neurons.

  SLM nonlinearity may be compensated by the input frame grabber.
  L1 and L2 fields are spatially interleaved. Weights are initialized using
  a sum of random outer-products.

  Trinary quantization of weight updates is an option.
  It is based on the method of Shoemaker, Carlin, and Shimabukuro.

  Thresholded output error depends only on correctness of output sign.

  The classification label is coded in a Hadamard vector instead of having
  ten outputs with each representing a class and choosing the maximum one. Instead,
  we calculate the distance of the output pattern from each of class-
  labeled patterns and choose the closest one. In other words, vector
  quantization is performed on the output patterns.

  Customized for Cohu 1122 CCD camera in variable integration mode.

  Yuri Owechko, Hughes Research Laboratories

  */

#include <graph.h>
#include <ctype.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <process.h>
#include <time.h>
#include <float.h>
#include <dos.h>
#include <search.h>
#include <stdlib.h>

 /* For VFG and IPA */
#include <sysvsp.h>
#include <gaoi.h>
#include <argtypes.h>
#include <fcb.h>
#include <ipa.h>
```

```
/* First bit controls laser and detector LC cells,
   second bit controls object LC cell, third bit controls
   phase of ref. */

#define WRITE_POS 3
#define WRITE_NEG 7
#define READ_POS 0
#define READ_NEG 4

#define SPONN_INPUT 1
#define SPONN_OUTPUT 2

#define NCLASS 10

int brand(),arand(),getbits(),round(),hsgn(),compare_sign();
void prepare_fg(),exemplar_digit(),init_weights(),verify_align();
void readout_L1(),readout_L2(),input_ex();
void exemplar_random_L0(),disp_cell(),transfer_L1();
void train_W1(),train_W2(),backward(),draw_digit();
void select_INPUT(),select_OUTPUT(),snap_xtrg();
float error_learn(),error_read(),read_neuron(),fsigmo(),ghump();
float dist2();

FILE *fopen(),*stream,*fp;

int xout[32][64][16],yout[32][64][16],xt[100],yt[100];
int xin_obj[32],yin_obj[64],mxmax,mymax,mmax,seed,seed2;
int xin_ref[32],yin_ref[64],size,size2,size9,npixneu;
int arraydim,arraydim2,base1,base2,xmax,ymax,dxi,dyi,dxo,dyo;
int confus_train[NCLASS][NCLASS],confus_test[NCLASS][NCLASS];
int pxout[32][64],pyout[32][64],gain,offsetval;
long int on_time,off_time,video_delay;
float sum_pos[32][32],sum_neg[32][32],neuron_norm[32][64];
float yy1[30][18],yy2[16][2],delta1[30][18],delta2[16][2],looptime;
float toterr[1000],toterr_read[1000],rms_err,totrms_err,fgamma_y1,fgamma_y2;

 /* Initialize N=32 Hadamard output vectors as labels for each of the NCLASS output classes */

    int iex[NCLASS][32]={
{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{ 1,-1, 1,-1, 1,-1, 1,-1, 1,-1, 1,-1, 1,-1, 1,-1, 1,-1, 1,-1, 1,-1, 1,-1, 1,-1, 1,-1, 1,-1, 1,-1},
{ 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1},
{ 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1, 1,-1,-1, 1},
{ 1, 1, 1, 1,-1,-1,-1,-1, 1, 1, 1, 1,-1,-1,-1,-1, 1, 1, 1, 1,-1,-1,-1,-1, 1, 1, 1, 1,-1,-1,-1,-1},
{ 1,-1, 1,-1,-1, 1,-1, 1, 1,-1, 1,-1,-1, 1,-1, 1, 1,-1, 1,-1,-1, 1,-1, 1, 1,-1, 1,-1,-1, 1,-1, 1},
{ 1, 1,-1,-1,-1,-1, 1, 1, 1, 1,-1,-1,-1,-1, 1, 1, 1, 1,-1,-1,-1,-1, 1, 1, 1, 1,-1,-1,-1,-1, 1, 1},
{ 1,-1,-1, 1,-1, 1, 1,-1, 1,-1,-1, 1,-1, 1, 1,-1, 1,-1,-1, 1,-1, 1, 1,-1, 1,-1,-1, 1,-1, 1, 1,-1},
{ 1, 1, 1, 1, 1, 1, 1, 1,-1,-1,-1,-1,-1,-1,-1,-1, 1, 1, 1, 1, 1, 1, 1, 1,-1,-1,-1,-1,-1,-1,-1,-1},
{ 1,-1, 1,-1, 1,-1, 1,-1,-1, 1,-1, 1,-1, 1,-1, 1, 1,-1, 1,-1, 1,-1, 1,-1,-1, 1,-1, 1,-1, 1,-1, 1}
            };

struct dosdate_t date;
struct dostime_t time1;
struct dostime_t time2;

char error_message[] = "This video mode is not supported";
char buffer[255],comment[200],comment2[200],comment3[200],stemp[200];
```

```
        main()
        {
        int n,m,mx,my,mx2,my2,i,j,k,nepoch,nepoch2,ntotal,kmax1,imax1,imax2;
        int err_value,nframes,nclass,nexemp,nexemptot,ntest,ntesttot,decquant,declut2;
        int xbase,ybase,zoomval,v1[256],dig_index,c,nterm,ncount,decdisp,declut;
        int i1,i2,nx,ny,value,shift,kx,ky,xindex,yindex,obj_bias;
        long int delay3,int_time,lval,klong;
        long int ltime,prevtime;
        float ftime,gain2,ontime,offtime,error,f1,f2,e1;
        float inttime,inttime2,fval,eps1,eps2,delta2gain;
        char blk,ch,cval;

        base1=592;  /* 592=250h PPI 1 base address */
        base2=596;  /* 596=254h PPI 2 base address */
        blk=219;    /* Solid block ASCII character */

/*      Set mode of DIO48 I/O card for Mode 0, all outputs  */

        outp(base1+3,128);
        outp(base2+3,128);

        if ((stream=fopen("learn34s.inp","r"))==NULL)
          {printf("cannot open learn34s.inp\n");
          exit(0);}
        else
          printf("learn34s.inp parameter file opened for reading\n");

        fgets(comment2,80,stream);
        fgets(comment,80,stream);
        fgets(comment3,80,stream);

        printf("Enter 1 for full graphics, 0 otherwise\n");
        fgets(stemp,200,stream);
        fscanf(stream,"%d\n",&decdisp);
        printf("%d\n",decdisp);

        printf("Enter no. of classes to learn, no. of exemplars per class,
        and no. of test patterns per class\nnclass*(nexemp+ntest)<=240\n");
        fgets(stemp,200,stream);
        fscanf(stream,"%d %d %d\n",&nclass,&nexemp,&ntest);
        printf("%d %d %d\n",nclass,nexemp,ntest);
        nexemptot=nclass*nexemp;
        ntesttot=nclass*ntest;

        printf("Enter zoom value for display field size : ");
        fgets(stemp,200,stream);
        fscanf(stream,"%d\n",&zoomval);
        printf("%d\n",zoomval);

        arraydim=pow(2,9-zoomval);
        arraydim2=arraydim/2;
        printf("Display field dimension is %d\n",arraydim);

        size=1; size2=2*size; size9=9*size;
```

```
printf("Enter no. of training epochs up to 1000: ");
fgets(stemp,200,stream);
fscanf(stream,"%d\n",&nepoch);
printf("%d\n",nepoch);

printf("Enter no. of readout epochs up to 100: ");
fgets(stemp,200,stream);
fscanf(stream,"%d\n",&nepoch2);
printf("%d\n",nepoch2);

printf("Enter exemplar integration time in sec: ");
fgets(stemp,200,stream);
fscanf(stream,"%f\n",&inttime);
printf("%f\n",inttime);

printf("Enter Output Frame Grabber gain from 1 to 4: ");
fgets(stemp,200,stream);
fscanf(stream,"%f\n",&gain2);
printf("%f\n",gain2);
gain=255*gain2/4;

printf("Enter Output Frame Grabber offset from 0 to 255: ");
fgets(stemp,200,stream);
fscanf(stream,"%d\n",&offsetval);
printf("%d\n",offsetval);

printf("Output Frame Grabber input LUT: 0 for unmodified, 1 for sqrt \n");
fgets(stemp,200,stream);
fscanf(stream,"%d\n",&declut);
printf("%d\n",declut);

printf("Input Frame Grabber output LUT: 0 for unmodified, 1 for linearized output\n");
fgets(stemp,200,stream);
fscanf(stream,"%d\n",&declut2);
printf("%d\n",declut2);

printf("Enter fsigmo gamma values for y1 and y2: ");
fgets(stemp,200,stream);
fscanf(stream,"%f %f\n",&fgamma_y1,&fgamma_y2);
printf("%f %f\n",fgamma_y1,fgamma_y2);

printf("Enter 1 for trinary quantization of weight updates, 0 otherwise:\n");
fgets(stemp,200,stream);
fscanf(stream,"%d\n",&decquant);
printf("%d\n",decquant);

printf("Enter eps1 for y1 quantization and eps2 for delta2 and delta1:\n");
fgets(stemp,200,stream);
fscanf(stream,"%f %f\n",&eps1,&eps2);
printf("%f %f\n",eps1,eps2);

printf("Enter SLM turn-on and turn-off times in sec: ");
fgets(stemp,200,stream);
fscanf(stream,"%f %f\n",&ontime,&offtime);
printf("%f %f\n",ontime,offtime);
```

```
    printf("Enter seed integer for random number generator: ");
    fgets(stemp,200,stream);
    fscanf(stream,"%d\n",&seed);
    printf("%d\n",seed);
    seed2=seed;

    printf("Enter delta2 gain factor for backward error propagation: ");
    fgets(stemp,200,stream);
    fscanf(stream,"%f\n",&delta2gain);
    printf("%f\n",delta2gain);

    printf("Enter number of terms in outer-product sum for initializing weights: ");
    fgets(stemp,200,stream);
    fscanf(stream,"%d\n",&nterm);
    printf("%d\n",nterm);

    printf("Enter object plane value for use as reference in coherent detection (0-255): ");
    fgets(stemp,200,stream);
    fscanf(stream,"%d\n",&obj_bias);
    printf("%d\n",obj_bias);

    fclose(stream);

    video_delay=70000;
    nframes=1;

/* Prepare frame grabbers */

    prepare_fg(zoomval,declut,declut2);

/* Store exemplars in frame grabber memory */

    exemplar_digit(nclass,nexemp,ntest,obj_bias);

/* Calibrate time base */

    time(&ltime);
    prevtime=ltime;
    for(klong=0;klong<2000000;klong++);
    time(&ltime);
    ftime=ltime-prevtime;
    looptime=ftime/2000000.0;
    int_time=inttime/looptime;
    on_time=ontime/looptime;
    off_time=offtime/looptime;

/* Verify alignment of CCD pixels for optical output */

    verify_align(obj_bias);

/* Initialize weights before learning */

    init_weights(int_time,nterm);

    for(j=0;j<=9;j++){
     for(i=0;i<=9;i++){
       confus_train[i][j]=0;}}
```

```
if (_setvideomode(_ERESCOLOR) == 0) {
        printf ("%s\n", error_message);
        exit(0);
        }

    sprintf(buffer, "learn34s.C: OPTICAL BACKPROPAGATION NEURAL NETWORK");
    _settextposition(1,0);
    _outtext(buffer);

    sprintf(buffer, "  DEMONSTRATION OF HANDWRITTEN DIGIT RECOGNITION");
    _settextposition(2,0);
    _outtext(buffer);

    sprintf(buffer,"1248 Neurons (676 Input, 540 Hidden, 32 Output) and 382,000 Weights");
    _settextposition(3,0);
    _outtext(buffer);

    sprintf(buffer,"Learning Phase");
    _settextposition(4,0);
    _outtext(buffer);

    sprintf(buffer,"Total Epoch Error:");
    _settextposition(5,0);
    _outtext(buffer);

    sprintf(buffer,"0");
    _settextposition(7,0);
    _outtext(buffer);

    sprintf(buffer,"1");
    _settextposition(7,39);
    _outtext(buffer);

    sprintf(buffer,"|");
    _settextposition(6,39);
    _outtext(buffer);

    sprintf(buffer,"|");
    _settextposition(6,0);
    _outtext(buffer);

    sprintf(buffer,"Epoch:");
    _settextposition(8,0);
    _outtext(buffer);

    sprintf(buffer,"Exemplar:");
    _settextposition(8,13);
    _outtext(buffer);

    sprintf(buffer,"Classification:");
    _settextposition(10,40);
    _outtext(buffer);

    dxi=234/26;
    dyi=130/26;
    dxo=60;
    dyo=4;
```

```
/* Borders for displays of input/output fields */

    sprintf(buffer, "Input Field L0");
    _settextposition(22,7);
    _outtext(buffer);

    sprintf(buffer, "Output Field L2");
    _settextposition(22,46);
    _outtext(buffer);

    _setcolor(12);
    _rectangle(_GBORDER,0,140,2+26*dxi,155+26*dyi);
    _rectangle(_GBORDER,159+26*dxi,149,161+26*dxi+dxo,151+32*dyo);

/*****************************************************************/
/*          Learning Loop Using Exemplars                  */
/*****************************************************************/

    _dos_gettime(&time1);
    f1=1.0/(nexemptot);
    f2=1.0/sqrt(16*nexemptot);

    if((stream=fopen("lrn34s.csv","w"))==NULL)
    {printf("cannot open lrn34s.csv\n");
    exit(0);}
    else
    printf("\nlrn34s.csv file opened for writing\n");

    fprintf(stream,"learn34s.c: Selected subset of neuron values during learning\n\n");
    fprintf(stream,"Exemplar No. 1: x1,,,,,,,,,,y1,,,,,,,,,,x2,,,,,y2,,,,,e1,,,,,,,,,,delta1
    ,,,,,,,,,,Exemplar No. 2: x1,,,,,,,,,,y1,,,,,,,,,,x2,,,,,y2,,,,,e1,,,,,,,,,,delta1,,,,,,,,,,Total RMS Error\n\n");

    for(n=0;n<nepoch;n++){

     for(j=0;j<=9;j++){
      for(i=0;i<=9;i++){
       confus_train[i][j]=0;}}

     sprintf(buffer,"%d  ",n);
     _settextposition(8,9);
     _outtext(buffer);
     sprintf(buffer,"Exemplar:    ");
     _settextposition(8,13);
     _outtext(buffer);

     totrms_err=0;
     error=0;
      for(c=0;c<nexemptot;c++){
        sprintf(buffer,"%d  ",c+1);
        _settextposition(8,26);
        _outtext(buffer);
        dig_index=fmod((double)c,(double)nclass);
        if(decdisp)
          draw_digit(c);
        input_ex(c); /* Input L0 exemplar */
        readout_L1(nframes,c);
        transfer_L1();
        readout_L2(nframes,c,obj_bias);
        err_value=error_learn(dig_index);
        error=error+err_value;
        totrms_err=totrms_err+rms_err;

        backward(nframes,delta2gain,c,obj_bias);

        train_W2(int_time,decquant,eps1,eps2);
        train_W1(int_time,decquant,eps2,c,obj_bias);

        }

      fval=f2*sqrt(totrms_err);
      fprintf(stream,",%f\n",fval);

      toterr[n]=f1*error;

      sprintf(buffer,"|");
      _settextposition(6,39);
      _outtext(buffer);

      sprintf(buffer,"|");
      _settextposition(6,0);
      _outtext(buffer);

     sprintf(buffer,"%2.3f",toterr[n]);
     _settextposition(5,20);
     _outtext(buffer);
     value=toterr[n]*40;

     for(i=0;i<value;i++){
       sprintf(buffer,"%c",blk);
       _settextposition(6,i);
       _outtext(buffer); }
     for(i=value;i<40;i++){
       sprintf(buffer," ");
       _settextposition(6,i);
       _outtext(buffer); }

      sprintf(buffer,"|");
      _settextposition(6,39);
      _outtext(buffer);

      sprintf(buffer,"|");
      _settextposition(6,0);
      _outtext(buffer);

  if(kbhit()){
   ch=getch();
   nepoch=n;
   break; }
  }

_dos_gettime(&time2);
```

```
/******************************************************************/
/*      Readout Loop Using Test Patterns                    */
/******************************************************************/

    fprintf(stream,"\n\nSelected subset of neuron values during readout\n\n");
    fprintf(stream,"Exemplar No. 1:
    x1,,,,,,,,,,,y1,,,,,,,,,,,x2,,,,,y2,,,,,Exemplar No. 2: x1,,,,,,,,,,,y1,,,,,,,,,,,x2,,,,,y2\n\n");

    sprintf(buffer,"Testing Phase   ");
    _settextposition(4,0);
    _outtext(buffer);
    sprintf(buffer,"Epoch:   ");
    _settextposition(8,0);
    _outtext(buffer);

    f1=1.0/(nclass*ntest);

    for(j=0;j<=9;j++){
     for(i=0;i<=9;i++){
      confus_test[i][j]=0;}}

    for(n=0;n<nepoch2;n++){
     sprintf(buffer,"%d  ",n);
     _settextposition(8,9);
     _outtext(buffer);
     sprintf(buffer,"Exemplar:    ");
     _settextposition(8,13);
     _outtext(buffer);

     error=0;
      for(c=nexemptot;c<(nexemptot+ntesttot);c++){
        sprintf(buffer,"%d  ",c-nexemptot+1);
        _settextposition(8,26);
        _outtext(buffer);
        dig_index=fmod((double)c,(double)nclass);
        if(decdisp)
         draw_digit(c);
        input_ex(c);
        readout_L1(nframes,c-nexemptot);
        transfer_L1();
        readout_L2(nframes,c-nexemptot,obj_bias);
        err_value=error_read(dig_index);
        error=error+err_value;
        }

     toterr_read[n]=f1*error;
     sprintf(buffer,"%2.3f",toterr_read[n]);
     _settextposition(5,20);
     _outtext(buffer);
     value=toterr_read[n]*40;

     for(i=0;i<value;i++){
      sprintf(buffer,"%c",blk);
      _settextposition(6,i);
      _outtext(buffer); }
     for(i=value;i<40;i++){
      sprintf(buffer," ");
      _settextposition(6,i);
      _outtext(buffer); }
      sprintf(buffer,"|");
      _settextposition(6,39);
      _outtext(buffer);

      sprintf(buffer,"|");
      _settextposition(6,0);
      _outtext(buffer);

    if(kbhit()){
      nepoch2=n;
      break; }
    }

  fclose(stream);

  outp(base1,READ_POS);
  disp_cell(2,0);

  select_OUTPUT();

  vfg_xtrigger(ON);
  _setvideomode(_DEFAULTMODE);  /* restore video mode */

  if((stream=fopen("learn34s.csv","w"))==NULL)
  {printf("cannot open learn34s.csv\n");
  exit(0);}
  else
  printf("\nlearn34s.csv file opened for writing\n");

  fprintf(stream,"learn34s.c Two-Layer Optical Backprop.
  With Coherent Detection (Handwritten digit recognition)\n");
  fprintf(stream,"Output class encoded in Hadamard output patterns; linearized SLM\n");
  fprintf(stream,"Crystal: %s",comment2);
  _dos_getdate(&date);
  fprintf(stream,"Date: %d-%d-%d ",date.month,date.day,date.year);
  fprintf(stream,"%s",comment);
  fprintf(stream,"%s",comment3);
  fprintf(stream,"No. of classes= %d\n",nclass);
  fprintf(stream,"No. of training and test patterns per class= \n%d, %d\n",nexemp,ntest);
  fprintf(stream,"zoomval= %d\n",zoomval);
  fprintf(stream,"Display field dimension is \n%d\n",arraydim);
  fprintf(stream,"xmax and ymax= \n%d, %d\n",xmax,ymax);
  fprintf(stream,"1248 total neurons: 676 unipolar units in L0; 540 bipolar units in L1;
  and 32 bipolar units in L2\n");
  fprintf(stream,"No. of weights= 382000\n");
  fprintf(stream,"No. of training and readout epochs: ");
  fprintf(stream,"%d; %d\n",nepoch,nepoch2);
  fprintf(stream,"Exemplar integration time in sec: ");
  fprintf(stream,"%f\n",inttime);
  fprintf(stream,"Output Frame Grabber gain: ");
  fprintf(stream,"%f\n",gain2);
  fprintf(stream,"Output Frame Grabber offset: ");
  fprintf(stream,"%d\n",offsetval);
  fprintf(stream,"Output Frame Grabber input LUT: 1 for square root; 0 for linear\n");
  fprintf(stream,"%d\n",declut);
  fprintf(stream,"Input Frame Grabber output LUT: 0 for unmodified; 1 for linearized SLM intensity\n");
  fprintf(stream,"%d\n",declut2);
  fprintf(stream,"Enter fsigmo gamma values for y1 and y2:\n");
```

```
    fprintf(stream,"%f %f\n",fgamma_y1,fgamma_y2);
    fprintf(stream,"Enter 1 for trinary quantization of weight updates; 0 otherwise\n");
    fprintf(stream,"%d\n",decquant);
    fprintf(stream,"Enter eps1 for y1 quantization; and eps2 for delta2 and delta1:\n");
    fprintf(stream,"%f, %f\n",eps1,eps2);
    fprintf(stream,"LCLV turn-on and turn-off delays in sec= \n%f, %f\n",ontime,offtime);
    fprintf(stream,"No. of samples per neuron\n%d\n",npixneu);
    fprintf(stream,"No. of video frames averaged\n%d\n",nframes);
    fprintf(stream,"Enter seed integer for random number generator\n");
    fprintf(stream,"%d\n",seed2);
    fprintf(stream,"Enter delta2 gain factor for backward error propagation\n");
    fprintf(stream,"%f\n",delta2gain);
    fprintf(stream,"Enter number of terms in gray-scale outer-product sum for initializing weights:\n");
    fprintf(stream,"%d\n",nterm);
    fprintf(stream,"Enter object plane bias value for coherent detection\n");
    fprintf(stream,"%d\n",obj_bias);
    fprintf(stream,"Learning start time: %d:%d:%d  Stop time: %d:%d:%d\n",time1.hour,
        time1.minute,time1.second,time2.hour,time2.minute,time2.second);

    fprintf(stream,"\n\nTraining set confusion matrix for 10 digits 0 to 9\n\n,,Input Digit\n\n");
    for(j=0;j<NCLASS;j++){
     for(i=0;i<NCLASS;i++){
       fprintf(stream,"%d, ",confus_train[i][j]);}
     fprintf(stream,"\n");}

    fprintf(stream,"\n\nTest set confusion matrix for 10 digits 0 to 9\n\n,,Input Digit\n\n");
    for(j=0;j<NCLASS;j++){
     for(i=0;i<NCLASS;i++){
       fprintf(stream,"%d, ",confus_test[i][j]);}
     fprintf(stream,"\n");}

    fprintf(stream,"\n\nLearning Error vs. Epoch\n\n");
    for(n=0;n<nepoch;n++){
     fprintf(stream,"%d, %2.3f\n",n,toterr[n]);}

   fprintf(stream,"\n\nReading Error vs. Epoch\n\n");
    for(n=0;n<nepoch2;n++){
     fprintf(stream,"%d, %2.3f\n",n,toterr_read[n]);}

    fclose(stream);

    }

/* Prepares frame grabbers */

    void prepare_fg(zoomval,declut,declut2)
    int zoomval,declut,declut2;
    {
    int v1[256],j;

/* Set up VFG boards. VFG 1 and VFG 2 are the INPUT and OUTPUT boards, respectively */

    if(load_cnf("c:\\visnplus\\lib\\user.cnf")==NO_ERROR)
     printf("Loaded configuration file\n");
    else
     printf("Could not load configuration file\n");
    initsys();
    err_level(2);

/* Set up VFG 2 */

    select_OUTPUT();
    vfg_level(gain,offsetval);
    vfg_setvframe(I);
    vfg_camera(VIDEO0);
    vfg_xtrigger(ON);

/*    Modify INPUT LUT of Output Frame Grabber  */

    if(declut){
     for(j=0;j<=255;j++)  {
       v1[j]=round(sqrt(255.)*sqrt(j));   }
     vfg_wlutseg(INPUT,0,0,256,v1);
     vfg_groupsel(GREEN); vfg_banksel(0); }

/* Set output LUTs of Output Frame Grabber for display.  */

    if(declut){
     for(j=0;j<=255;j++){
       v1[j]=round(pow(j,2)/255.);  }
     v1[128]=0;
     vfg_wlutseg(RED,0,0,256,v1);
     v1[0]=255;
     v1[255]=0;
     vfg_wlutseg(BLUE,0,0,256,v1);
     v1[0]=0;
     v1[128]=255;
     vfg_wlutseg(GREEN,0,0,256,v1);
     vfg_groupsel(GREEN); vfg_banksel(0); }
    else {
     for(j=0;j<=255;j++){
       v1[j]=j;  }
     v1[128]=0;
     vfg_wlutseg(RED,0,0,256,v1);
     v1[0]=255;
     v1[255]=0;
     vfg_wlutseg(BLUE,0,0,256,v1);
     v1[0]=0;
     v1[128]=255;
     vfg_wlutseg(GREEN,0,0,256,v1);
     vfg_groupsel(GREEN); vfg_banksel(0); }

    vfg_xtrigger(ON);
```

```
/* Set up Input Frame Grabber */

     switch(declut2)
       {
       case 0:
         for(j=0;j<=255;j++){
           v1[j]=j;}
         break;
       case 1:
         if((stream=fopen("linear.lut","r"))==NULL)
              {printf("cannot open linear.lut\n");
              exit(0);}
         else
              printf("\nlinear.lut file opened for reading\n");
         for(j=0;j<=255;j++){
           fscanf(stream,"%d\n",&v1[j]);}
         fclose(stream);
         break;
       }

      select_INPUT();
      vfg_wlutseg(GREEN,0,0,256,v1);     /* Adjust Input Frame Grabber output LUT */
      vfg_wlutseg(RED,0,0,256,v1);
      v1[254]=0;
      v1[255]=0;
      vfg_wlutseg(BLUE,0,0,256,v1);
      vfg_setvframe(I);
      vfg_aclear(0,0);
      vfg_zoom(zoomval);
      disp_cell(0,0);
      }


/* Store exemplars in frame grabber memory */

      void exemplar_digit(nclass,nexemp,ntest,obj_bias)
      int nclass,nexemp,ntest,obj_bias;
      {
      int i,j,c,xbase,ybase,xindex,yindex,kx,ky,nx,ny,dig_index;
      long int offset;
      double fracpart,intpart;
      char ch,cval;

/*     Read in exemplars for digits 0 to 9 from bitmap file  */

      if ((fp=fopen("bt1430.bmp","rb"))==NULL)
        {printf("cannot open bt1430.bmp\n");
        exit(0);}
      else
        printf("Reading exemplars from bt1430.bmp and writing to VFG 1\n");
```

```
 /* Store exemplars in 64x64 subframes (0,1) to (15,15) */

     c=0;
     for(j=0;j<55;j++){
       for(i=0;i<40;i+=NCLASS){
         for(dig_index=0;dig_index<nclass;dig_index++){
           fracpart=modf((double)c/16.0,&intpart);
           xindex=c-16*intpart;
           yindex=intpart+1;
           xbase=xindex*arraydim;
           ybase=yindex*arraydim;
           vfg_block(0,xbase,ybase,30,18,obj_bias);
           kx=0; ky=0;

           for(ny=0;ny<26;ny++){
             offset=0x76L+520L*j*26+ny*520L+(i+dig_index)*13;
             fseek(fp,offset,SEEK_SET);

             for(nx=0;nx<13;nx++){
               cval=fgetc(fp);
               ch=getbits(cval,7,4);
               if(ch==0)
                 vfg_block(0,xbase+arraydim2+kx,ybase+ky,size,size,255);
               else
                 vfg_block(0,xbase+arraydim2+kx,ybase+ky,size,size,0);
               kx++;
               if(kx==30){
                 kx=0; ky++;}

               ch=getbits(cval,3,4);
               if(ch==0)
                 vfg_block(0,xbase+arraydim2+kx,ybase+ky,size,size,255);
               else
                 vfg_block(0,xbase+arraydim2+kx,ybase+ky,size,size,0);
               kx++;
               if(kx==30){
                 kx=0; ky++;}
                         }}
            c++;
            printf("%d ",c);
            if(c==nclass*(nexemp+ntest))break;}
          if(c==nclass*(nexemp+ntest))break; }
        if(c==nclass*(nexemp+ntest))break; }

     printf("\n");
     fclose(fp);

    }

/* Verifies alignment of CCD pixels for optical output */

   void verify_align(obj_bias)
     int obj_bias;
     {
     int i,j,n,xbase,ybase,shift,mx,my;
     char ch;
     long lval,klong;
 /* Read in output coordinates and normalization factors from align*.dat
   generated using alignvfg.c.  */

      if ((stream=fopen("align64.dat","r"))==NULL)
        {printf("\ncannot open align64.dat\n");
        exit(0);}
      else
        printf("\nalign64.dat coordinate file opened for reading\n");

      fscanf(stream,"%d %d %d\n",&xmax,&ymax,&npixneu);

      for(j=0;j<ymax;j++){
        for(i=0;i<xmax;i++){
          for(n=0;n<npixneu;n++){
            fscanf(stream,"%d %d ",&xout[i][j][n],&yout[i][j][n]);}
          fscanf(stream,"%f",&neuron_norm[i][j]);
          fscanf(stream,"\n");}}

      fclose(stream);


      outp(base1,READ_POS);
      select_INPUT();

/* Generate SPONN input coordinate vectors */

      for(i=0;i<xmax;i++){
        xin_ref[i]=arraydim2+i*size;}
      for(i=0;i<ymax;i++){
        yin_ref[i]=i*size;}

      for(i=0;i<xmax;i++){
        xin_obj[i]=i*size;}
      for(i=0;i<ymax;i++){
        yin_obj[i]=i*size;}

/* (0,0) and (1,0) are positive and negative working areas;
   ref-on in (2,0); object subsets A and B in (3,0) and (4,0), respectively.
   ref-on and bias-on in object plane in (7,0). */

      xbase=2*arraydim;
      ybase=0;
      vfg_block(0,xbase+arraydim2,ybase,arraydim2,arraydim,255);
      xbase=7*arraydim;
      vfg_block(0,xbase+arraydim2,ybase,arraydim2,arraydim,255);
      vfg_block(0,xbase,ybase,arraydim2,arraydim,obj_bias);

      xbase=3*arraydim;
      ybase=0;

      shift=1;
      for(j=0;j<ymax;j++){
        shift=1-shift;
        for(i=0;i<xmax;i+=2){
          vfg_block(0,xbase+xin_obj[i+shift],ybase+yin_obj[j],size,size,255);
          }}
```

```
     xbase=4*arraydim;
     ybase=0;

     shift=0;
     for(j=0;j<ymax;j++){
       shift=1-shift;
       for(i=0;i<xmax;i+=2){
         vfg_block(0,xbase+xin_obj[i+shift],ybase+yin_obj[j],size,size,255);
         }}

     select_OUTPUT();
     vfg_xtrigger(ON);

/* Verify alignment of output pixels relative to optical neurons. */

     select_INPUT();
     disp_cell(0,0);

/* Turn on every other input neuron in L1 object plane (L0 to L1 weights) */

     shift=1;
     for(my=0;my<18;my++){
       shift=1-shift;
       for(mx=0;mx<15;mx++){
         vfg_block(0,xin_obj[2*mx+shift],yin_obj[my],size,size,255);}}

/* L2 object */
     for(mx=0;mx<8;mx++){
       vfg_block(0,xin_obj[4*mx],yin_obj[23],size2,size9,255);}

/* L1 object */
     shift=0;
     for(my=0;my<9;my++){
       shift=1-shift;
       for(mx=0;mx<15;mx++){
         vfg_block(0,xin_obj[2*mx+shift],yin_obj[my+32],size,size,255);}}

/* L2 object */
     for(mx=0;mx<8;mx++){
       vfg_block(0,xin_obj[4*mx],yin_obj[41],size2,size9,255);}

/* L1 object */
     shift=0;
     for(my=9;my<18;my++){
       shift=1-shift;
       for(mx=0;mx<15;mx++){
         vfg_block(0,xin_obj[2*mx+shift],yin_obj[my+41],size,size,255);}}

     select_OUTPUT();

     lval=3*on_time;
     for(klong=0;klong<lval;klong++);
     snap_xtrg(video_delay);

/* Color sampled CCD pixels */
/* L1 object (L0->L1) */
    for(my=0;my<18;my++){
     for(mx=0;mx<30;mx++){
      for(i=0;i<npixneu;i++){
        vfg_wpixel(0,xout[mx][my][i],yout[mx][my][i],255);}}}

/* L2 object */
    for(my=0;my<2;my++){
     for(mx=0;mx<16;mx++){
      for(i=0;i<npixneu;i++){
        vfg_wpixel(0,xout[1+2*mx][27+18*my][i],yout[2*mx][27+18*my][i],255);}}}

/* L1 object (L2->L1) */
    for(my=0;my<9;my++){
     for(mx=0;mx<30;mx++){
      for(i=0;i<npixneu;i++){
        vfg_wpixel(0,xout[mx][32+my][i],yout[mx][32+my][i],255);}}}
    for(my=9;my<18;my++){
     for(mx=0;mx<30;mx++){
      for(i=0;i<npixneu;i++){
        vfg_wpixel(0,xout[mx][41+my][i],yout[mx][41+my][i],255);}}}

    printf("\nOutput pixels marked in red on monitor\n");

    printf("Hit any key to continue\n");
    for(;;)if(kbhit())break;
    ch=getch();

    select_INPUT();
    vfg_block(0,0,0,arraydim2,arraydim,0);

    }

/* Initialize weights using sum of random 8-bit gray-scale outer-products */

    void init_weights(int_time,nterm)
    long int_time;
    int nterm;
    {
    int ncount,n,xbase,ybase,mx,my,value,i,j,flag;
    long lval,klong;
    char ch;

    printf("\nInitializing weights with sum of random bipolar outer-products...\n\nHit any key to begin learning\n");

    lval=4*int_time;
    ncount=0;
    flag=1;

    for(;;){

    if(ncount==nterm){
     ncount=0;
     flag=1; }
```

```
/* (L0)(L1) Weight initialization vectors in (5,0). */

    ybase=0;
    xbase=5*arraydim;
    srand(seed+16000+ncount);
   /* L0 */
     for(my=0;my<23;my++){
      for(mx=0;mx<30;mx++){
        vfg_block(0,xin_ref[mx]+xbase,yin_ref[my]+ybase,size,size,arand());
                 }}
   /* L1 */
     for(my=0;my<18;my++){
      for(mx=0;mx<30;mx++){
       value=arand();
        vfg_block(0,xin_obj[mx]+xbase,yin_obj[my]+ybase,size,size,value);
                 }}

/* (L1)(L2) vectors for weight initialization in (6,0). */

    xbase=6*arraydim;
    srand(seed+20000+ncount);
     /* L1 */
     for(my=0;my<9;my++){
      for(mx=0;mx<30;mx++){
       value=arand();
        vfg_block(0,xin_ref[mx]+xbase,yin_ref[my+23]+ybase,size,size,value);
        vfg_block(0,xin_obj[mx]+xbase,yin_obj[my+32]+ybase,size,size,value);
                 }}

     for(my=0;my<9;my++){
      for(mx=0;mx<30;mx++){
       value=arand();
        vfg_block(0,xin_ref[mx]+xbase,yin_ref[my+41]+ybase,size,size,value);
        vfg_block(0,xin_obj[mx]+xbase,yin_obj[my+50]+ybase,size,size,value);
                 }}

     /* L2 */
      for(mx=0;mx<16;mx++){
       value=arand();
        vfg_block(0,xin_ref[2*mx]+xbase,yin_ref[32]+ybase,size2,size9,value);
        vfg_block(0,xin_obj[2*mx]+xbase,yin_obj[23]+ybase,size2,size9,value);
               }

      for(mx=0;mx<16;mx++){
       value=arand();
        vfg_block(0,xin_ref[2*mx]+xbase,yin_ref[50]+ybase,size2,size9,value);
        vfg_block(0,xin_obj[2*mx]+xbase,yin_obj[41]+ybase,size2,size9,value);
                }
     if(flag)
      outp(base1,WRITE_POS);
     else
      outp(base1,WRITE_NEG);

     disp_cell(5,0);
     for(klong=0;klong<=lval;klong++);

     select_OUTPUT();
     snap_xtrg(video_delay);
     select_INPUT();
```

```
      disp_cell(6,0);
      for(klong=0;klong<=int_time;klong++);

      select_OUTPUT();
      snap_xtrg(video_delay);
      select_INPUT();


      if(kbhit())break;

      ncount++;
      flag=1-flag;
      }

    ch=getch();
    outp(base1,READ_POS);

    }

/* Snaps image when Cohu camera is in variable integration mode */

    void snap_xtrg(video_delay)
     long video_delay;
    {
     long jlong;

     vfg_xtrigger(ON);
     for(jlong=0;jlong<video_delay;jlong++);
     vfg_xtrigger(OFF);
     for(jlong=0;jlong<video_delay;jlong++);
    }

/* This routine reads out neuron (mx,my) by averaging over pixels */

    float read_neuron(mx,my)
     int mx,my;
     {
     int j,k;
     long jlong;
     float fval;

     fval=0;
     for(j=0;j<npixneu;j++){
      fval+=vfg_brpixel(0,xout[mx][my][j],yout[mx][my][j]);
      }

     fval=fval/npixneu;

     return(fval);
     }

/* Draws Exemplar No. c on the computer monitor */

    void draw_digit(c)
    int c;
    {
    int xindex,yindex,xbase,ybase,i,j,nx,ny,val;
    double fracpart,intpart;
```

```
     fracpart=modf((double)c/16.0,&intpart);
     xindex=c-16*intpart;
     yindex=intpart+1;
     xbase=xindex*arraydim+arraydim2;
     ybase=yindex*arraydim;

     ny=25; nx=0;
     for(j=0;j<23;j++){
       for(i=0;i<30;i++){
         val=vfg_rpixel(0,xbase+i,ybase+j);
         if(val)
           _setcolor(15);
         else
           _setcolor(0);
         _rectangle(_GFILLINTERIOR,1+nx*dxi,148+ny*dyi,1+(nx+1)*dxi,148+(ny+1)*dyi);
         nx++;
         if(nx==26){nx=0; ny--;}
       }}

     }

/* Inputs Exemplar No. c */

     void input_ex(c)
     int c;
     {
     int xindex,yindex;
     double fracpart,intpart;

     fracpart=modf((double)c/16.0,&intpart);
     xindex=c-16*intpart;
     yindex=intpart+1;

     disp_cell(xindex,yindex);

     }

/*  This subroutine displays cell (m,n) */

     void disp_cell(m,n)
     int m,n;
     {
     int xbase,ybase;

     xbase=m*arraydim;
     ybase=n*arraydim;

     vfg_roam(xbase,ybase);

     }

/* This function calculates L1 values */

     void readout_L1(nframes,c)
     int nframes,c;
     {
     int i1,i2,mx2,my2,val,i,j;
     long int klong;
     float fval;
```

```
  select_OUTPUT();

  for(my2=0;my2<18;my2++){
   for(mx2=0;mx2<30;mx2++){
    sum_pos[mx2][my2]=0;
    sum_neg[mx2][my2]=0;
    }}

  outp(base1,READ_POS);
  for(klong=0;klong<on_time;klong++);
  for(i2=0;i2<nframes;i2++){
   snap_xtrg(video_delay);
   for(my2=0;my2<18;my2++){
    for(mx2=0;mx2<30;mx2++){
     sum_pos[mx2][my2]+=read_neuron(mx2,my2);}}
               }

  outp(base1,READ_NEG);
  for(klong=0;klong<on_time;klong++);
  for(i2=0;i2<nframes;i2++){
   snap_xtrg(video_delay);
   for(my2=0;my2<18;my2++){
    for(mx2=0;mx2<30;mx2++){
     sum_neg[mx2][my2]+=read_neuron(mx2,my2);}}
               }

 fval=1.0/nframes;
 for(my2=0;my2<18;my2++){
  for(mx2=0;mx2<30;mx2++){
   yy1[mx2][my2]=fsigmo(fval*(sum_pos[mx2][my2]-sum_neg[mx2][my2]),fgamma_y1);}}

     if(c<2) {
      for(i=0;i<30;i+=6){
       for(j=0;j<18;j+=9){
        fprintf(stream,"%2.3f,",fval*(sum_pos[mx2][my2]-sum_neg[mx2][my2]));}}

      fprintf(stream,",");

      for(i=0;i<30;i+=6){
       for(j=0;j<18;j+=9){
        fprintf(stream,"%2.3f,",yy1[i][j]);}}
      fprintf(stream,",");
          }

 select_INPUT();

 }

/* Transfer L1 output to L1 input */

 void transfer_L1()
 {
 int mx2,my2,pixval;

 for(my2=0;my2<9;my2++){
  for(mx2=0;mx2<30;mx2++){
   pixval=255*yy1[mx2][my2];
   vfg_block(0,xin_ref[mx2],yin_ref[my2+23],size,size,pixval);
   }}
```

```
    for(my2=9;my2<18;my2++){
     for(mx2=0;mx2<30;mx2++){
      pixval=255*yy1[mx2][my2];
      vfg_block(0,xin_ref[mx2],yin_ref[my2+32],size,size,pixval);
      }}

    }

/* This function reads and calculates L2 values and displays them on the
  computer monitor.  */

    void readout_L2(nframes,c,obj_bias)
    int nframes,c,obj_bias;
    {
    int i1,i2,mx2,my2,val,xval,yval,k,c1,c2,i,j;
    long int klong;
    float fval;

    vfg_block(0,0,23,32,9,obj_bias);
    vfg_block(0,0,41,32,9,obj_bias);
    disp_cell(0,0);

    select_OUTPUT();

    for(my2=0;my2<2;my2++){
     for(mx2=0;mx2<16;mx2++){
      sum_pos[mx2][my2]=0;
      sum_neg[mx2][my2]=0;
      }}

    outp(base1,READ_POS);
    for(klong=0;klong<on_time;klong++);
    for(i2=0;i2<nframes;i2++){
     snap_xtrg(video_delay);
     for(my2=0;my2<2;my2++){
      for(mx2=0;mx2<16;mx2++){
       sum_pos[mx2][my2]+=read_neuron(1+2*mx2,27+18*my2);}}
                }

    outp(base1,READ_NEG);
    for(klong=0;klong<on_time;klong++);
    for(i2=0;i2<nframes;i2++){
     snap_xtrg(video_delay);
     for(my2=0;my2<2;my2++){
      for(mx2=0;mx2<16;mx2++){
       sum_neg[mx2][my2]+=read_neuron(1+2*mx2,27+18*my2);}}
                }

    c1=160+26*dxi+dxo/2; c2=150+dyo/2;
    _setcolor(0);
    _rectangle(_GFILLINTERIOR,160+26*dxi,150,160+26*dxi+dxo,150+32*dyo);

    k=0;
    fval=1/nframes;
    for(my2=0;my2<2;my2++){
     for(mx2=0;mx2<16;mx2++){
      yy2[mx2][my2]=fsigmo(fval*(sum_pos[mx2][my2]-sum_neg[mx2][my2]),fgamma_y2);
      fval=0.5*fabs(yy2[mx2][my2]);
```

```
       xval=dxo*fval; yval=dyo*fval;
       if(yy2[mx2][my2]>0)
        _setcolor(15);
       else
        _setcolor(13);
       _rectangle(_GFILLINTERIOR,c1-xval,c2+k*dyo-yval,c1+xval,c2+k*dyo+yval);
       k++;
       }}

         if(c<2) {

          for(mx2=0;mx2<16;mx2+=4){
           fprintf(stream,"%f, ",fval*(sum_pos[mx2][0]-sum_neg[mx2][0]));}

          fprintf(stream,",");

          for(mx2=0;mx2<16;mx2+=4){
           fprintf(stream,"%2.3f,",yy2[mx2][0]);}

          fprintf(stream,",");

               }

    select_INPUT();

    vfg_block(0,0,23,arraydim,9,0);
    vfg_block(0,0,41,arraydim,9,0);

    }

/* Calculates L2 output error
  in learning phase */

    float error_learn(dig_index)
    int dig_index;
    {
    int j,mx2,my2,class,c;
    float e2,error,mindist,fval;

    j=0;
    fval=0;
    for(my2=0;my2<2;my2++){
     for(mx2=0;mx2<16;mx2++){
      e2=0.5*(iex[dig_index][j]-yy2[mx2][my2]);
      fval=fval+e2*e2;
      delta2[mx2][my2]=e2*ghump(yy2[mx2][my2]);
      j++;}}

    rms_err=fval;

    mindist=1000.;

    for(c=0;c<NCLASS;c++){
     fval=dist2(c);
      if(fval<mindist){
       mindist=fval;
       class=c;}
             }
     confus_train[dig_index][class]++;

     if(class==dig_index){
      error=0.;
      _setcolor(_GREEN);
      sprintf(buffer,"Correct  ");
      _settextposition(10,56);
      _outtext(buffer);
      _setcolor(_WHITE);}
     else{
      error=1.;
      _setcolor(_RED);
      sprintf(buffer,"Incorrect");
      _settextposition(10,56);
      _outtext(buffer);
      _setcolor(_WHITE);}

     return(error);

     }

/* Calculates distance squared between current value of output vector yy2
  and exemplar output pattern c */

     float dist2(c)
     int c;
     {
     int mx2,my2,j;
     float fval;

     j=0;
     fval=0;
     for(my2=0;my2<2;my2++){
      for(mx2=0;mx2<16;mx2++){
       fval+=pow(yy2[mx2][my2]-iex[c][j],2);
       j++;}}

     return(fval);

     }

/* Calculates L2 output error for exemplar c in reading phase */

     float error_read(dig_index)
     int dig_index;
     {
     int mx2,my2,class,c;
     float e2,mindist,error,fval;

     mindist=1000.;

/* Choose output class */

     for(c=0;c<NCLASS;c++){
      fval=dist2(c);
       if(fval<=mindist){
        mindist=fval;
        class=c;}
             }
     confus_test[dig_index][class]++;

     if(class==dig_index){
      error=0;
      _settextcolor(_GREEN);
      sprintf(buffer,"Correct  ");
      _settextposition(10,56);
      _outtext(buffer);
      _settextcolor(_WHITE);}
     else{
      error=1;
      _settextcolor(_RED);
      sprintf(buffer,"Incorrect");
      _settextposition(10,56);
      _outtext(buffer);
      _settextcolor(_WHITE);}

     return(error);

     }

/* Backpropagate the error signal. */

     void backward(nframes,delta2gain,c,obj_bias)
     int nframes,c,obj_bias;
     float delta2gain;
     {
     long int klong;
     int mx2,my2,i2,pixval;
     float fval,e1;

    /* Bias for coherent readout of L1 */

     vfg_block(0,0,32,32,9,obj_bias);
     vfg_block(0,0,50,32,9,obj_bias);
     vfg_block(0,arraydim,32,32,9,obj_bias);
     vfg_block(0,arraydim,50,32,9,obj_bias);

     /* Write delta2 to L2 reference */

     for(my2=0;my2<2;my2++){
      for(mx2=0;mx2<16;mx2++){
       pixval=delta2gain*255*delta2[mx2][my2];
       if(pixval>0)
        vfg_block(0,xin_ref[2*mx2],yin_ref[18*my2+32],size2,size9,pixval);
       else
        vfg_block(0,arraydim+xin_ref[2*mx2],yin_ref[18*my2+32],size2,size9,-pixval);
       }}

     /* Read L1 output and calculate delta1 */


    select_OUTPUT();

    for(my2=0;my2<18;my2++){
     for(mx2=0;mx2<10;mx2++){
      sum_pos[mx2][my2]=0;
      sum_neg[mx2][my2]=0;}}
```

```
 outp(base1,READ_POS);
 for(klong=0;klong<on_time;klong++);

 for(i2=0;i2<nframes;i2++){
  snap_xtrg(video_delay);
  for(my2=0;my2<9;my2++){
   for(mx2=0;mx2<30;mx2++){
    sum_pos[mx2][my2]+=read_neuron(mx2,32+my2);}}

  for(my2=9;my2<18;my2++){
   for(mx2=0;mx2<30;mx2++){
    sum_pos[mx2][my2]+=read_neuron(mx2,41+my2);}}
              }
 select_INPUT();
 disp_cell(1,0);

 select_OUTPUT();
 outp(base1,READ_NEG);
 for(klong=0;klong<on_time;klong++);

 for(i2=0;i2<nframes;i2++){
  snap_xtrg(video_delay);
  for(my2=0;my2<9;my2++){
   for(mx2=0;mx2<30;mx2++){
    sum_neg[mx2][my2]+=read_neuron(mx2,32+my2);}}

  for(my2=9;my2<18;my2++){
   for(mx2=0;mx2<30;mx2++){
    sum_neg[mx2][my2]+=read_neuron(mx2,41+my2);}}
              }

fval=1.0/(255*nframes);
for(my2=0;my2<18;my2++){
 for(mx2=0;mx2<30;mx2++){
  e1=-fval*(sum_pos[mx2][my2]-sum_neg[mx2][my2]);
  delta1[mx2][my2]=ghump(yy1[mx2][my2])*e1;}}

    if(c<2) {
     for(my2=0;my2<18;my2+=9){
      for(mx2=0;mx2<30;mx2+=10){
       e1=-fval*(sum_pos[mx2][my2]-sum_neg[mx2][my2]);
       fprintf(stream,"%f,",e1);}}

     fprintf(stream,",");

     for(my2=0;my2<18;my2+=9){
      for(mx2=0;mx2<30;mx2+=10){
       fprintf(stream,"%f,",delta1[mx2][my2]);}}
```

```
          fprintf(stream,",");
               }

     select_INPUT();


     vfg_block(0,0,32,arraydim,9,0);
     vfg_block(0,0,50,arraydim,9,0);
     vfg_block(0,arraydim,32,arraydim,9,0);
     vfg_block(0,arraydim,50,arraydim,9,0);

     }

/* This function updates the weights between L2 and L1  */

     void train_W2(int_time,decquant,eps1,eps2)
     int decquant;
     float eps1,eps2;
     long int int_time;
     {
     int mx2,my2,pixval;
     long int klong;

      /* Write delta2 to L2 reference */

     for(my2=0;my2<2;my2++){
      for(mx2=0;mx2<16;mx2++){
       if(decquant)
         pixval=255*hsgn(delta2[mx2][my2],eps2);
       else
         pixval=255*delta2[mx2][my2];

       if(pixval>=0){
         vfg_block(0,xin_ref[2*mx2],yin_ref[18*my2+32],size2,size9,pixval);
         vfg_block(0,9*arraydim+xin_ref[2*mx2],yin_ref[18*my2+32],size2,size9,pixval);}
       else{
         vfg_block(0,arraydim+xin_ref[2*mx2],yin_ref[18*my2+32],size2,size9,-pixval);
         vfg_block(0,8*arraydim+xin_ref[2*mx2],yin_ref[18*my2+32],size2,size9,-pixval);
          }

      }}

    /* Write y1 to L1 object */

    for(my2=0;my2<9;my2++){
     for(mx2=0;mx2<30;mx2++){
      if(decquant)
        pixval=255*hsgn(yy1[mx2][my2],eps1);
      else
        pixval=255*yy1[mx2][my2];

      if(pixval>=0){
        vfg_block(0,xin_obj[mx2],yin_obj[my2+32],size,size,pixval);
        vfg_block(0,arraydim+xin_obj[mx2],yin_obj[my2+32],size,size,pixval);}
      else{
        vfg_block(0,9*arraydim+xin_obj[mx2],yin_obj[my2+32],size,size,-pixval);
        vfg_block(0,8*arraydim+xin_obj[mx2],yin_obj[my2+32],size,size,-pixval);}
     }}
```

```
for(my2=9;my2<18;my2++){
 for(mx2=0;mx2<30;mx2++){
  if(decquant)
   pixval=255*hsgn(yy1[mx2][my2],eps1);
  else
   pixval=255*yy1[mx2][my2];

  if(pixval>=0){
   vfg_block(0,xin_obj[mx2],yin_obj[my2+41],size,size,pixval);
   vfg_block(0,arraydim+xin_obj[mx2],yin_obj[my2+41],size,size,pixval);}
  else{
   vfg_block(0,9*arraydim+xin_obj[mx2],yin_obj[my2+41],size,size,-pixval);
   vfg_block(0,8*arraydim+xin_obj[mx2],yin_obj[my2+41],size,size,-pixval);}
 }}


/* Write delta2 to L2 object */

for(my2=0;my2<2;my2++){
 for(mx2=0;mx2<16;mx2++){
  if(decquant)
   pixval=255*hsgn(delta2[mx2][my2],eps2);
  else
   pixval=255*delta2[mx2][my2];

  if(pixval>=0){
   vfg_block(0,xin_obj[2*mx2],yin_obj[18*my2+23],size2,size9,pixval);
   vfg_block(0,9*arraydim+xin_obj[2*mx2],yin_obj[18*my2+23],size2,size9,pixval);}
  else{
   vfg_block(0,arraydim+xin_obj[2*mx2],yin_obj[18*my2+23],size2,size9,-pixval);
   vfg_block(0,8*arraydim+xin_obj[2*mx2],yin_obj[18*my2+23],size2,size9,-pixval);
    }
 }}

/* Write y1 to L1 reference */

for(my2=0;my2<9;my2++){
 for(mx2=0;mx2<30;mx2++){
  if(decquant)
   pixval=255*hsgn(yy1[mx2][my2],eps1);
  else
   pixval=255*yy1[mx2][my2];

  if(pixval>=0){
   vfg_block(0,xin_ref[mx2],yin_ref[my2+23],size,size,pixval);
   vfg_block(0,arraydim+xin_ref[mx2],yin_ref[my2+23],size,size,pixval);}
  else{
   vfg_block(0,8*arraydim+xin_ref[mx2],yin_ref[my2+23],size,size,-pixval);
   vfg_block(0,9*arraydim+xin_ref[mx2],yin_ref[my2+23],size,size,-pixval);}
 }}

for(my2=9;my2<18;my2++){
 for(mx2=0;mx2<30;mx2++){
  if(decquant)
   pixval=255*hsgn(yy1[mx2][my2],eps1);
  else
   pixval=255*yy1[mx2][my2];
```

```
        if(pixval>=0){
          vfg_block(0,xin_ref[mx2],yin_ref[my2+32],size,size,pixval);
          vfg_block(0,arraydim+xin_ref[mx2],yin_ref[my2+32],size,size,pixval);}
        else{
          vfg_block(0,8*arraydim+xin_ref[mx2],yin_ref[my2+32],size,size,-pixval);
          vfg_block(0,9*arraydim+xin_ref[mx2],yin_ref[my2+32],size,size,-pixval);}
      }}

/* Adjust W2 weights in 4 phases to account for ++, --,+-, and -+ combinations */

     disp_cell(0,0); /* ++ */
     outp(base1,WRITE_POS);
     for(klong=0;klong<=int_time;klong++);
     disp_cell(8,0); /* -- */
     for(klong=0;klong<=int_time;klong++);
     disp_cell(1,0); /* -+ */
     outp(base1,WRITE_NEG);
     for(klong=0;klong<=int_time;klong++);
     disp_cell(9,0); /* +- */
     for(klong=0;klong<=int_time;klong++);
     outp(base1,READ_POS);

     vfg_block(0,0,23,arraydim,36,0);
     vfg_block(0,arraydim,23,arraydim,36,0);
     vfg_block(0,8*arraydim,23,arraydim,36,0);
     vfg_block(0,9*arraydim,23,arraydim,36,0);
       }

/* This function updates the weights between L0 and L1  */

     void train_W1(int_time,decquant,eps2,c,obj_bias)
     long int int_time;
     int c,decquant,obj_bias;
     float eps2;
     {
     int mx2,my2,xbase,ybase,pixval,xindex,yindex;
     long int klong;
     double fracpart,intpart;

     fracpart=modf((double)c/16.0,&intpart);
     xindex=c-16*intpart;
     yindex=intpart+1;

     /* Display L0 exemplar */

     disp_cell(xindex,yindex);

     /* Write delta1 to L1 object */

     xbase=xindex*arraydim;
     ybase=yindex*arraydim;

     /* Positive adjustment of L0-L1 weights */

     for(my2=0;my2<18;my2++){
      for(mx2=0;mx2<30;mx2++){
        if(decquant)
          pixval=255*hsgn(delta1[mx2][my2],eps2);
      else
       pixval=255*delta1[mx2][my2];
      if(pixval>=0)
       vfg_block(0,xbase+xin_obj[mx2],ybase+yin_obj[my2],size,size,pixval);
      else
       vfg_block(0,xbase+xin_obj[mx2],ybase+yin_obj[my2],size,size,0);
      }}

    outp(base1,WRITE_POS);
    for(klong=0;klong<=int_time;klong++);


    /* Negative adjustment of L0-L1 weights */

    for(my2=0;my2<18;my2++){
     for(mx2=0;mx2<30;mx2++){
      if(decquant)
       pixval=255*hsgn(delta1[mx2][my2],eps2);
      else
       pixval=255*delta1[mx2][my2];
      if(pixval<0)
       vfg_block(0,xbase+xin_obj[mx2],ybase+yin_obj[my2],size,size,-pixval);
      else
       vfg_block(0,xbase+xin_obj[mx2],ybase+yin_obj[my2],size,size,0);
      }}

    outp(base1,WRITE_NEG);
    for(klong=0;klong<=int_time;klong++);
    outp(base1,READ_POS);

    vfg_block(0,xbase,ybase,30,18,obj_bias);

      }


/* Output layer neuron sigmoidal activation function (range is [-1,1]) */

    float fsigmo(fsum,gamma)
    float fsum,gamma;
    {
    float fval;
    fval=2.0/(1+exp(-gamma*fsum))-1;
    return(fval);
    }


/* Output layer backpropagation function  */

    float ghump(fsum)
    float fsum;
    {
    float fval;
    fval=(1.0-fsum)*(1.0+fsum);
    return(fval);
    }
```

```
/* Trinary quantization function with outputs (-1,0,1) */

    int hsgn(net,eps)
    float net,eps;
    {
    int ival;

    if(net<-eps)
      ival=-1;
    else if(net>eps)
      ival=1;
    else
      ival=0;

    return(ival);

    }


/* Returns 1 or -1 randomly */

    int brand()
    {
    return(2*getbits(rand(),0,1)-1);
    }

/* Returns random integer uniformly distributed between 0 and 255 */

    int arand()
    {
    int ival;

    ival=255.0*rand()/RAND_MAX;
    return(ival);
    }

/*  This function gets n bits to the right of and including position p
   in integer x.  */

    getbits(x,p,n)
    unsigned x,p,n;
    {
    return((x>>(p+1-n))&~(~0<<n));
    }

/* Returns 0 if arguments have same sign, 1 otherwise */

    int compare_sign(m,fn)
    int m;
    float fn;
    {
    int val;

    if(m*fn>=0)
      val=0;
    else
      val=1;
```

```
     return(val);

     }

/*  This function rounds a floating point number to the nearest integer */

     int round(fval)
      float fval;
      {
      int val;
      float remain;

      remain=fval-floor(fval);
      if(remain>=0.5)
      val=ceil(fval);
      else
      val=floor(fval);

      return(val);
      }

/* Selects OUTPUT VFG board */

     void select_OUTPUT()
     {

     select_vfg(SPONN_INPUT);
     write_reg(vfg[MAPEN],0);
     vfg_memen(OFF);
     select_vfg(SPONN_OUTPUT);
     write_reg(vfg[MAPEN],1);
     vfg_memen(ON);

     }

/* Selects INPUT VFG board */

     void select_INPUT()
     {

     select_vfg(SPONN_OUTPUT);
     write_reg(vfg[MAPEN],0);
     vfg_memen(OFF);
     select_vfg(SPONN_INPUT);
     write_reg(vfg[MAPEN],1);
     vfg_memen(ON);

     }
```

Appendix B
PUBLICATIONS AND PRESENTATIONS

B.1 Publications

Y. Owechko, "Applications of Liquid Crystals in Image and Signal Processing," in *A Guide to Liquid Crystal Research*, P. J. Collings and J. S. Patel, eds., to be published in 1996 by Oxford University Press.

Y. Owechko, "Opportunities for Optoelectronics in Holographic Data Storage and Neural Networks," *in Opportunities for Innovation: Optoelectronics,* A. Goutzoulis, ed., NIST publication NIST GCR 95-672, 1995.

Y. Owechko and B. H. Soffer, "Holographic Neurocomputer Utilizing Laser-Diode Light Source," SPIE Proceedings Vol. 2565, (1995).

Y. Owechko and B. H. Soffer, "Holographic Neural Networks Based on Multi-Grating Processes," in *Real-Time Optical Information Processing*, B. Javidi and J. Horner, eds., Academic Press (1994).

Y. Owechko, "Applications of Spatial Light Modulators in Optical Processing," in *Spatial Light Modulators: Materials, Devices, and Applications,* U. Efron, ed., Marcel Dekker (1994).

Y. Owechko, "Optical Neural Networks Based on Real-Time Holography," Proceedings of International Topical Conference on Research Trends in Nonlinear and Quantum Optics, La Jolla, 1994.

Y. Owechko, "Cascaded-Grating Holography for Artificial Neural Networks," Applied Optics **32**, 1380-1398 (1993).

Y. Owechko and B. H. Soffer, "Optical Neurocomputer Based on Multiple-Grating Holography," Proceedings of Government Microcircuit Applications Conference, New Orleans, 1993.

Y. Owechko and B. H. Soffer, "Holographic Neurocomputer for Backpropagation Based on Cascaded-Grating Holography," SPIE Proceedings Vol. 2026, (1993).

Y. Owechko and B. H. Soffer, "Implementation of Neural Networks Based on Photorefractive Beam Fanning," Proceedings of Topical Meeting on Photorefractive Materials, Effects, and Devices, Kiev, Ukraine, 1993.

Y. Owechko, "Optical Implementation of Backpropagation Neural Networks Using Cascaded-Grating Holography," International Journal of Optical Computing **2**, 201-231, (1991) (This issue was published in 1993). (Invited Paper)

B. H. Soffer and Y. Owechko, "Recent Advances in Optical Neural Networks," Proceedings of Conference "From Galileo's 'Occhialino' to Optoelectronics", Padov University, June 9-12, 1992, World Scientific. (Invited Paper)

Y. Owechko and B. H. Soffer, “A Programmable Optical Neuro-Computer Based on Photorefractive Holograms,” Proceedings of Government Microcircuit Applications Conference, Las Vegas, 1992.

Y. Owechko and B. H. Soffer, “Multi-Layer Optical Neural Networks,” SPIE Proceedings Vol 1773, 1992.

Y. Owechko and B. H. Soffer, “An Optical Interconnection Method for Neural Networks Using Self-Pumped Phase Conjugate Mirrors,” Optics Letters 16, 675-677 (1991).

G. J. Dunning, Y. Owechko, and B.H. Soffer, “Hybrid Opto-Electronic Neural Networks Using a Mutually-Pumped Phase Conjugate Mirror,” Optics Letters 16, 928-930(1991).

Y. Owechko and B. H. Soffer, “Optical Neural Networks Based on Liquid Crystal Light Valves and Photorefractive Crystals,” Proceedings of SPIE/SPSE Symposium on Electronic Imaging Science and Technology, San Jose, Feb. 1991, Vol. 1455, 136-144. (Invited Paper)

B.H. Soffer, Y. Owechko, and G.J. Dunning, “A Photorefractive Optical Neural Network,” SPIE Proceedings Vol. 1347, 1, (1990). (Invited Paper)

B.H. Soffer, Y. Owechko, and G.J. Dunning, “A Photorefractive Optical Neural Network,” Proceedings of 15th Congress of International Commission for Optics, Aug. 1990, Garmish, Germany.

B.2 Presentations

B. H. Soffer and Y. Owechko, “An Optical Neural Network Based on Distributed Holographic Gratings,” II Reunion Iberoamericana de Optica, Guanajuato, Mexico, 1995. (Invited Talk)

Y. Owechko and B. H. Soffer, “An Optical Neural Network Based on Distributed Holographic Gratings for ATR”, International Conference on Neural Networks, Perth, 1995.

Y. Owechko and B. H. Soffer, “Holographic Neurocomputer Utilizing Laser-Diode Light Source,” SPIE Conference on Optical Implementation of Information Processing, San Diego, 1995.

Y. Owechko, “Optical Neural Networks Based on Real-Time Holography,” International Topical Conference on Research Trends in Nonlinear and Quantum Optics, La Jolla, 1993. (Invited Talk)

Y. Owechko, “Holographic Neural Networks: A Systems Perspective,” OSA Topical Meeting on Optical Computing, Palm Springs, 1993. (Invited Talk)

Y. Owechko and B. H. Soffer, “Optical Neurocomputer Based on Multiple-Grating Holography,” Government Microcircuit Applications Conference, New Orleans, 1993.

Y. Owechko, “Programmable Optical Neural Networks,” OSA 1992 Annual Meeting, Albuquerque, 1992. (Invited Talk)

Y. Owechko and B. H. Soffer, “A Programmable Optical Neuro-Computer Based on Photorefractive Holograms,” Government Microcircuit Applications Conference, Las Vegas, 1992.

Y. Owechko, Lake Louis conference on electronic and optical implementations of neural networks, 1992.

Y. Owechko and B. H. Soffer, "Multi-Layer Optical Neural Networks," SPIE Conference 1773 on Photonics for Computers, Neural Networks, and Memories, San Diego, 1992.

B. H. Soffer and Y. Owechko, "Recent Advances in Optical Neural Networks," Galilean Year Celebration Conference, Padova, Italy, 1992. (Invited Talk)

Y. Owechko and B. H. Soffer, "Recent Advances in Optical Neural Networks Using Beam Fanning in Photorefractive Crystals," LEOS '91, San Jose, Nov. 1991. (Invited Talk)

Y. Owechko and B. H. Soffer, "Optical Neural Networks Based on Liquid Crystal Light Valves and Photorefractive Crystals," SPIE/SPSE Symposium on Electronic Imaging Science and Technology, San Jose, Feb. 1991. (Invited Talk)

Y. Owechko, G. Dunning, and B. Soffer, "Optical Neural Networks Based on Stimulated Photorefractive Effects," OSA 1990 Annual Meeting, Boston. (Invited Talk)

Y. Owechko, "Holographic Neural Networks," 1990 International Topical Meeting on Optical Computing, Kobe, Japan. (Invited Talk)

Y. Owechko, B.H. Soffer, and G.J. Dunning, "Photorefractive Optical Neural Networks," International Joint Conference on Neural Networks, Washington, D.C., 1990. (Invited Talk)

G.J. Dunning and Y. Owechko, "Multi-Port Optical Neural Network Using a Mutually Pumped Phase Conjugate Mirror," IEEE Meeting on Nonlinear Optics: Materials, Phenomena, and Devices, Kuau, 1990.

G.J. Dunning, Y. Owechko, and B.H. Soffer, "Optical Neural Network Using a Mutually Pumped Phase Conjugate Mirror," OSA Annual Meeting, Orlando, 1989.

Y. Owechko, "Self-Pumped Optical Neural Networks," OSA Topical Meeting on Optical Computing, Salt Lake City, 1989.

Y. Owechko, "Stimulated Photorefractive Optical Neural Networks," Third Annual Parallel Processing Symposium, Cal. State Fullerton, 1989. (Invited Talk)